Data Understanding, Data Analysis, and Data Science Course Notes

Volume 3: Spotlight on Machine Learning

Patrick Boily

July 2024

Quadrangle | Idlewyld Analytics and Consulting Services

This one goes out to the "Welsh" contingent: Elowyn, Llewellyn, and Gwynneth. Your world is going to be a whole lot different than mine was; maybe data can even help make some of it better. But one thing's for sure: data is not going away any time soon – better be prepared.

Series Preface

The *first* thing to know about *Data Understanding, Data Analysis, and Data Science* (DUDADS) is that it isn't really a "book". It makes more sense to think of it as **course notes**, or as a **reference manual** and a source of examples and application.

I borrow some of its contents from authors who do a better job of explaining things than I could hope to do; I also sometimes modify their examples and code to better suit my pedagogical needs.* Major influences include [1, 2, 3, 4, 5, 6, 8] – be sure to give these masterful works the attention they deserve!

The *second* thing to know about DUDADS is that it isn't really "a" book. It makes more sense to think of it as **a bunch of books in a trench coat, masquerading as single one**.† No one is expected to traverse DUDADS in one sitting, or even to tackle more than a few of its assigned chapters, sections, subsections, exercises at any given time; rather, it is intended to be read in parallel with guided lectures.

The *third* thing to know about DUDADS is that the practical examples use `R` and/or `Python`, for no particular reason other than that *some* programming language had to be used to illustrate the concepts. In the text, `R` code appears in blue boxes:

```
... some R code ...
```

Whereas `Python` code appears in green boxes:

```
... some Python code ...
```

You may look at some piece of code and think to yourself: "This isn't how I would do it" or "such-and-such a task would be easier to accomplish if we used module/package ABC or programming language XYZ". That's quite possible.

But finding the optimal tool is not the point of this book. In the first place, new data science tools appear regularly, and it would be a fool's errand to try to continuously modify the book to keep up with them.‡ In the second place, I am serious about the "Understanding" part of *Data Understanding, Data Analysis, and Data Science*, and that is why I favour a **tool-agnostic** approach.

---

* In all cases, I have attempted to properly cite and give credit where it is due. Get in touch if you find omissions!
† I paid heed to this realization by splitting it into a number of volumes.
‡ I am not saying that I won't be adding examples in different languages in the future, but let's not get ahead of ourselves.

The *fourth* thing to know about DUDADS is that it is not a place to go to in order to obtain a detailed step-by-step guide on "how to solve it". In person, my answer to a vast array of data science related questions is, rather anti-climatically: "it depends". Of course, it depends; on the data, on the objectives, on the cost associated with making a mistake, on the stakeholder's appetite for uncertainty, and, perhaps more surprisingly, on the analytical and data preparation choices that are made along the way.

To some, this might smack of post-modernism: "you are saying that there is no truth, and that data analysis is pointless!" To which I respond: "analysts have agency (lots of it, it turns out), and their choices *DO* influence the results, so make sure to run multiple analyses to determine the variability of the outcomes". That is the nature of the discipline.

The *last* thing you should probably know about DUDADS is that I have made a concerted effort to focus mainly on the **story** of (learning) data analysis and data science; sometimes, that comes at the expense of rigorous exposition.

> "The early stages of education have to include a lot of lies-to-children, because early explanations have to be simple. However, we live in a complex world, and lies-to-children must **eventually be replaced** by more complex stories if they are not to become delayed-action genuine lies." [7]

Some of the concepts and notions that I present are **incomplete** by design, but remain (I hope) true-to-their-spirit, or at least true "enough" for a first pass.[§] My position is that learning is an iterative process and that important take-aways from an early stage might need to be modified to account for new developments at a later date. But all things in good time: flexibility is a friend in your learning adventure; perfectionism, not always so.

Patrick Boily
Wakefield, July 2024

The DUDADS reference manuals are available at idlewyldanalytics.com

- Volume 1: ***Prelude to Data Understanding***
- Volume 2: ***Fundamentals of Data Insight***
- Volume 3: ***Spotlight on Machine Learning***
- Volume 4: ***Techniques of Data Analysis***
- Volume 5: ***Special Topics in Data Science and Artificial Intelligence***
- ***The Practice of Data Visualization*** (with S. Davies and J. Schellinck)

[§] In the parlance of the field, let me simply say that some of the details are left as an exercise for the reader (and can also be found in the numerous references).

Preface References

[1] C.C. Aggarwal. *Data Mining: the Textbook*. Cham: Springer, 2015.

[2] C.C. Aggarwal, ed. *Data Classification: Algorithms and Applications*. CRC Press, 2015.

[3] C.C. Aggarwal and C.K. Reddy, eds. *Data Clustering: Algorithms and Applications*. CRC Press, 2014.

[4] D. Dalpiaz. *R for Statistical Learning*. 2020.

[5] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, *2nd ed.* Springer, 2008.

[6] G. James et al. *An Introduction to Statistical Learning: With Applications in R*. Springer, 2014.

[7] I. Stewart, J. Cohen, and T. Pratchett. *The Science Of Discworld*. Ebury Publishing, 2002.

[8] H. Wickham and G. Grolemund. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly, Jan. 2017.

Contributors and Influences

A reference manual of this size could not have been compiled without the help of a multitude of individuals over the years, who provided contributions, influences, and/or inspiration:

Shintaro Hagiwara *Focus on Classification*

Olivier Leduc *Focus on Classification, Feature Selection and Dimension Reduction*

Andrew Macfie *Feature Selection and Dimension Reduction*

Aditya Maheshwari *Focus on Clustering, Feature Selection and Dimension Reduction*

Maia Pelletier *Feature Selection and Dimension Reduction*

Tristan Shaeen general editing

Jen Schellinck *Focus on Clustering*

A hearty "thank you" to everyone, and to all others with whom I have crossed paths on this data adventure!

Learning Paths

I mostly use the material found in this volume at various levels in my teaching at the University of Ottawa in the *Department of Mathematics and Statistics*, or for data workshops offered to professionals through *Idlewyld Analytics and Consulting Services*, the *Data Action Lab*, and the University of Ottawa's *Professional Development Institute*.

In particular, here is what I cover in various courses/workshops:

- **MAT3373** (*Methods of Machine Learning*) – Chapters 20, 21, 22, 23;
- **Introduction to Machine Learning** – Chapter 19.

The contents of Chapters 1-7 (DUDADS, volume 1) are pre-requisites for MAT3373.

Contents

List of Figures

List of Tables

Introduction to Machine Learning

19

by **Patrick Boily**

Data scientists are often introduced to the field *via* machine learning concepts, algorithms and applications, which we introduce in this chapter.

In Chapters 20, 21, and 22, we will discuss other technical aspects of machine learning, as well as more sophisticated algorithms (abundant details can also be found in [3, 1, 24, 6, 27], among others).

19.1 Preliminaries

> "Data is not information, information is not knowledge, knowledge is not understanding, understanding is not wisdom." (attributed to C. Stoll [30])

One of the challenges of working in the **data science** (DS), **machine learning** (ML), and **artificial intelligence** (AI) fields is that nearly all quantitative work can be described with some combination of the terms DS/ML/AI (often to a ridiculous extent). As such, it can be difficult to differentiate the discipline from other quantitative fields, which makes studying and learning it properly harder than it should.

Robinson [50] suggests that their relationships follow an inclusive **hierarchical** structure:

- in a first stage, **DS** provides "insights" *via* visualization and (manual) inferential analysis;
- in a second stage, **ML** yields "predictions" (or "advice"), while reducing the operator's analytical, inferential and decisional workload (although it is still present to some extent), and
- in the final stage, **AI** removes the need for oversight, allowing for automatic "actions" to be taken by a completely unattended system.

The goals of AI are laudable in an academic setting, but in practice, we believe that stakeholders should not seek to give up their agency in the decision-making process; as such, we follow the lead of various thinkers and suggest further splitting AI into "general AI" (which we will not be pursuing further at this stage) and "**augmented** intelligence".[1]

With this in mind, our definition of the DS/ML/AI approach is that it consists of quantitative processes[2] that can help users **learn actionable insights** about their situation without completely abdicating their decision-making responsibility.

In this chapter, we will take a brief look at:

1: Which can be viewed as ML "on steroids".

2: What H. Mason has called "the **working intersection** of statistics, engineering, computer science, domain expertise, and "hacking" [60].

- the **fundamentals** of data science (see Section 19.2);
- **association rules** mining (see Section 19.3);
- **supervised learning** and **classification**, with a focus on **decision trees** (see Section 19.4);
- **unsupervised learning** and **clustering**, with a focus on $k-$**means** (see Section 19.5), and
- some of the common **issues** and **challenges** encountered during the data science and machine learning process (see Section 19.6).

19.2 Statistical Learning

"We learn from failure, not from success!" (B. Stoker, *Dracula*)

As humans, we learn (at all stages) by first taking in our environment, and then by:

- answering questions about it;
- testing hypotheses;
- creating concepts;
- making predictions;
- creating categories, and
- classifying and grouping its various objects and attributes.

In a way, the main concept of DS/ML/AI is to try to teach our machines (and thus, ultimately, ourselves) to glean insight from data, and how to do this properly and efficiently, free of biases and pre-conceived notions – in other words, **can we design algorithms that can learn?**[3]

3: Note that this is not the same thing as asking whether we *should* design such algorithms.

In that context, the simplest DS/ML/AI method is **exploring the data** (or a representative sample) to:

- provide a summary through basic statistics – mean, variance, histograms, etc.;
- make its multi-dimensional structure evident through data visualization, and
- look for consistency, considering what is in there and what is missing.

19.2.1 Types of Learning

In the **statistical learning** context,[4] more sophisticated approaches traditionally fall into a **supervised** or an **unsupervised** learning framework.

4: A term sometimes used to describe general DS/ML/AI approaches, for no particular reason other than letting the audience know that the user has a background in mathematics and statistics.

Supervised learning is akin to "learning with a teacher." Typical tasks include **classification**, **regression**, **rankings**, and **recommendations**.

In supervised learning, algorithms use **labeled training data** to build (or train) a predictive model;[5] each algorithm's performance is evaluated using **test data** for which the label is known but not used in the prediction.[6]

5: For instance, students may need to answer each exam question based on what they learned from worked-out examples provided by the teacher/textbook.

6: The teacher provides the correct answers and marks the exam questions using the key, to continue the example.

In supervised learning, there are fixed **targets** against which to train the model (such as age categories, or plant species) – the categories (and their number) are known prior to the analysis.

Unsupervised learning, on the other hand, is akin to "self-learning by grouping similar exercises together as a study guide." Typical tasks include **clustering**, **association rules discovery**, **link profiling**, and **anomaly detection**. Unsupervised algorithms use **unlabeled data** to find natural patterns in the data; the drawback is that accuracy **cannot be evaluated** with the same degree of satisfaction.[7]

In unsupervised learning, we don't know what the target is, or even if there is one – we are simply looking for **natural groups** in the data.[8]

Other Learning Frameworks Some data science techniques fit into both camps; others can be either supervised or unsupervised, depending on how they are applied, but there are other conceptual approaches, especially for AI tasks:

- **semi-supervised learning** in which some data points have labels but most do not, which may occur when acquiring data is costly;[9]
- **reinforcement learning**, where an agent attempts to collect as much (short-term) reward as possible while minimizing (long-term) regret.[10]

19.2.2 DS and ML Tasks

Outside of academia, DS/ML/AI methods are only really interesting when they help ask and answer useful questions. Compare, for instance:

- **Analytics** – "How many clicks did this link get?"
- **Data Science** – "Based on the previous history of clicks on links of this publisher's site, can I predict how many people from Manitoba will read this specific page in the next three hours?" or "Is there a relationship between the history of clicks on links and the number of people from Manitoba who will read this specific page?"
- **Quantitative Methods** – "We have no similar pages whose history could be consulted to make a prediction, but we have reasons to believe that the number of hits will be strongly correlated with the temperature in Winnipeg. Using the weather forecast over the next week, can we predict how many people will access the specific page during that period?"

Data science and machine learning models are usually **predictive** (not **explanatory**): they show connections, and exploit correlations to make predictions, but they don't reveal why such connections exist.

Quantitative methods, on the other hand, usually assume a certain level of causal understanding based on various **first principles**. That distinction is not always understood properly by clients and consultants alike.

Common data science tasks include [46]:

- **classification** and **probability estimation** – which undergraduates are likely to succeed at the graduate level?
- **value estimation** – how much is a given client going to spend at a restaurant?
- **similarity matching** – which prospective clients are most similar to a company's established best clients?

7: The teacher would not be involved in the discovery process, say, and the students might end up with different groupings, as an example.

8: Perhaps junior students who like literature, have longish hair, and know how to cook *vs.* students who are on a sports team and have siblings *vs.* financial professionals with a penchant for superhero movies, craft beer and Hello Kitty backpack *vs.* ...

9: The teacher could provide worked-out examples and a list of unsolved problems to try out; the students try to find similar groups of unsolved problems and compare them with the solved problems to find close matches.

10: Embarking on a Ph.D. with an advisor, with all of the highs and the lows (and **maybe** a diploma at the end of the process?).

- **clustering** – do signals from a sensor form natural groups?
- **association rules discovery** – what books are commonly purchased together at an online retailer?
- **profiling** and **behaviour description** – what is the typical cell phone usage of a certain customer segment?
- **link prediction** – J. and K. have 20 friends in common: perhaps they'd be great friends?

A classic example is provided by the UCI Machine Learning Repository *Mushroom Dataset* [15]. Consider *Amanita muscaria* (commonly known as the fly agaric), a specimen of which is shown below.

Figure 19.1: *Amanita muscaria* (fly agaric), in the wild. Does it look dangerous to you?

Is it **edible**, or **poisonous**? There is a simple way to get an answer – eat it, wait, and see: if you do not die or get sick upon ingestion, it was **edible**; otherwise it was **poisonous**.

But this test in unappealing for various reasons, however. Apart from the obvious risk of death, we might not learn much from the experiment; it is possible that this specific specimen was poisonous due to some mutation or some other factor (or that the ingester had a pre-existing condition which combined with the fungus to cause discomfort, etc.), and that fly agaric is actually edible in general (unlikely, but not impossible).

A predictive model, which would use features of a vast collection of mushroom species and specimens (including their class) could help shed light on the matter: what do poisonous mushrooms have in common? What properties do edible mushrooms share?[11]

11: Note that this is not the same as understanding **why** a mushroom is poisonous or edible – the data alone cannot provide an answer to that question.

For instance, let's say that *Amanita muscaria* has the following features:

- **habitat**: woods;
- **gill size**: narrow;
- **spores**: white;
- **odor**: none,
- **cap color**: red.

We do not know **a priori** whether it is poisonous or edible. Is the available information sufficient to answer the question? Not on its own, no.[12]

12: A mycologist could perhaps deduce the answer from these features alone, but she would be using her experience with fungi to make a prediction, and so would not be looking at the features in a *vacuum*.

But we could use **past data**, with correct **edible** or **poisonous** labels and the **same set of predictors** to build various supervised **classification** models to attempt to answer the question.

A simple form of such model, a **decision tree**, is shown in Figure 19.2.

The model prediction for *Amanita muscaria* follows the **decision path** shown in Figure 19.3.

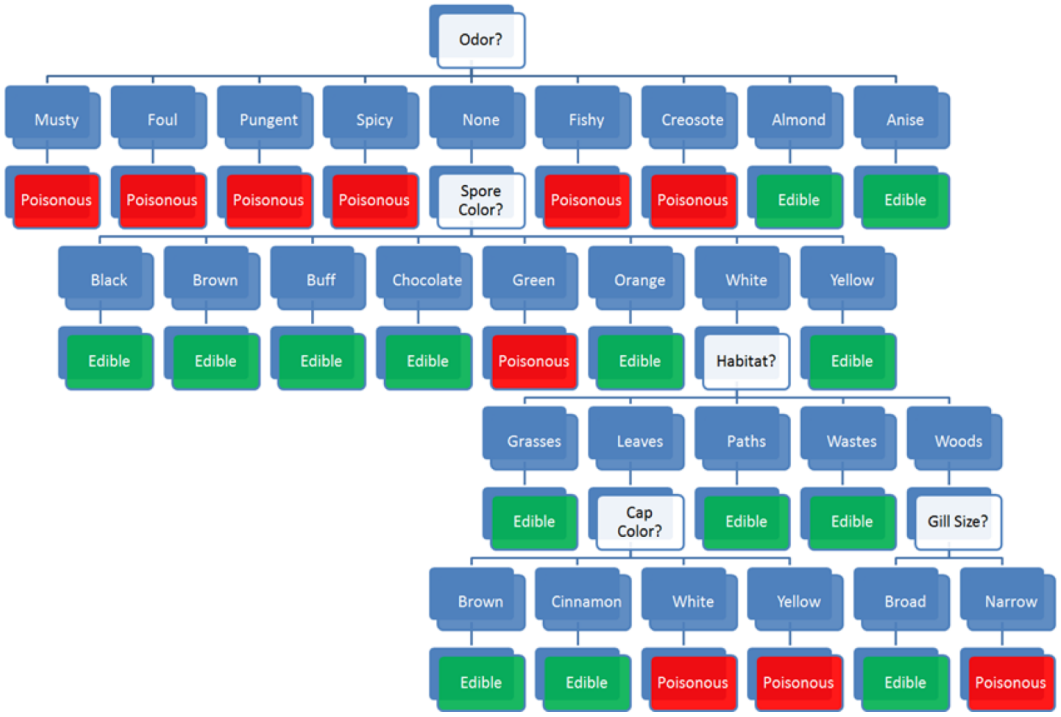


Figure 19.2: Decision tree for the mushroom classification problem [author unknown].

1. some mushroom **odors** (musty, spicy, etc.) are associated with poisonous mushrooms, some (almond, anise) with edible mushrooms, but there are mushrooms with no specific odor in either category – for mushroom with 'no odor' (as with *Amanita muscaria*), odor does not provide enough information for proper classification and we need to incorporate additional features into the decision path;
2. among mushrooms with no specific odor, some **spore colours** (black, etc.) are associated with edible mushrooms, some (almond, anise) with poisonous mushrooms, but there are mushrooms with 'white' spores in either category – the combination 'no odor and white spores' does not provide enough information to classify *Amanita muscaria* and we need to incorporate additional features into the decision path;
3. among mushrooms of no specific odor with white spores, some **habitats** (grasses, paths, wastes) are associated with edible mushrooms, but there are mushrooms in either category that are found in the 'woods' – the combination 'no odor, white spores, found in the woods' does not provide enough information to classify *Amanita muscaria* and we need to incorporate additional features into the decision path,
4. among white-spored forest mushroom with no specific odor, a broad **gill size** is associated with edible mushrooms, whereas a 'narrow' gill size is associated with poisonous mushrooms – as *Amanita muscaria* is a narrow-gilled, white-spored forest mushroom with no specific odor, the decision path predicts that it is **poisonous**.

Note that the **cap color** does not affect the decision path, however.[13] But the decision tree model **does not explain why** this particular combinations of features is associated with poisonous mushrooms – the decision path is not **causal**.

13: It would have had *Amanita muscaria*'s habitat been 'leaves'.

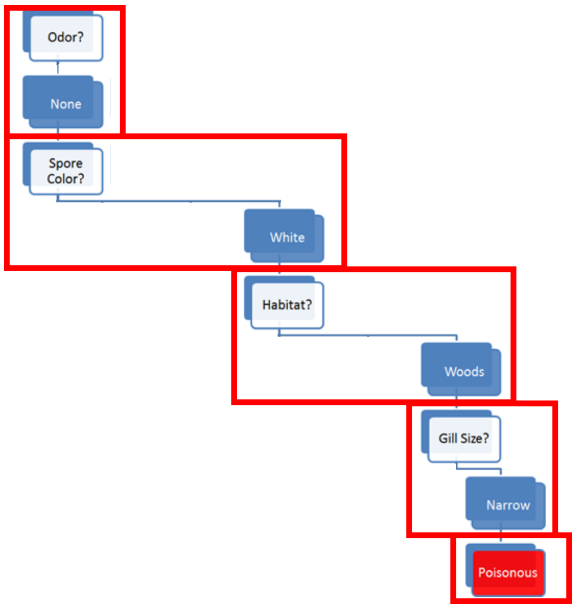


Figure 19.3: Decision path for *Amanita muscaria*.

At this point, a number of questions naturally arise:

- Would you have trusted an **edible** prediction?
- How are the features measured?
- What is the true cost of making a mistake?
- Is the data on which the model is built representative?
- What data is required to build trustworthy models?
- What do we need to know about the model in order to **trust it**?

The stage has now been set: this mushroom classification problem has all the hallmarks of a ML problem. Keep it in mind as a representative of the discipline in the sections that follow.

19.3 Association Rules Mining

"Correlation isn't causation. But it's a big hint." (E. Tufte)

19.3.1 Overview

Association rules discovery is a type of unsupervised learning that finds **connections** among the attributes (variables) and levels (values), and combinations thereof, of a dataset's observations. For instance, we might analyze a (hypothetical) dataset on the physical activities and purchasing habits of North Americans and discover that

- runners who are also triathletes (the **premise**) tend to drive Subarus, drink microbrews, and use smart phones (the **conclusion**), or
- individuals who have purchased home gym equipment are unlikely to be using it 1 year later, say.

But the presence of a **correlation** between the premise and the conclusion does not necessarily imply the existence of a **causal relationship** between them. It is rather difficult to "demonstrate" causation *via* data analysis; in

practice, decision-makers pragmatically (and often erroneously) focus on the second half of Tufte's rejoinder, which basically asserts that "there's no smoke without fire."

Case in point, while being a triathlete does not cause one to drive a Subaru, Subaru Canada thinks that the connection is strong enough to offer to reimburse the registration fee at an IRONMAN 70.3 competition (since at least 2018)! [8]

Market Basket Analysis Association rules discovery is also known as **market basket analysis** after its original application, in which supermarkets record the contents of shopping carts (the **baskets**) at check-outs to determine which items are frequently purchased together.

For instance, while bread and milk might often be purchased together, that is unlikely to be of interest to supermarkets given the frequency of market baskets containing milk **or** bread.[14]

14: In the mathematical sense of "or": one, or the other, or both.

Knowing that a customer has purchased bread does provide some information regarding whether they also purchased milk, but the individual probability that each item is found, separately, in the basket is so high to begin with that this insight is unlikely to be useful.

If 70% of baskets contain milk and 90% contain bread, say, we would expect **at least**

$$90\% \times 70\% = 63\%$$

of all baskets to contain milk **and** bread, should the presence of one in the basket be **totally independent** of the presence of the other.

If we then observe that 69% of baskets contain both items (a 1.10-fold increase on the expected proportion, assuming there is no link), we would conclude that there was at best a **weak correlation** between the purchase of milk and the purchase of bread.

Sausages and hot dog buns, on the other hand, which we might suspect are not purchased as frequently as milk and bread, might still be purchased as a pair more often than one would expect given the frequency of baskets containing sausages **or** buns.

If 10% of baskets contain sausages, and 5% contain buns, say, we would expect that

$$10\% \times 5\% = 0.5\%$$

of all baskets would contain sausages **and** buns, should the presence of one in the basket be **totally independent** of the presence of the other.

If we then observe that 4% of baskets contain both items (an 8-fold increase on the expected proportion, assuming there is no link), we would obviously conclude that there is a **strong correlation** between the purchase of sausages and the purchase of hot dog buns.

It is not too difficult to see how this information could potentially be used to help supermarkets turn a profit: announcing or advertising a sale on sausages while **simultaneously** (and quietly) raising the price of buns could have the effect of bringing in a higher number of customers into the store, increasing the sale volume for both items while keeping the combined price of the two items constant.[15]

15: The marketing team is banking on the fact that customers are unlikely to shop around to get the best deal on hot dogs AND buns, which may or may not be a valid assumption.

A (possibly) apocryphal story shows the limitations of association rules: a supermarket found an association rule linking the purchase of beer and diapers and consequently moved its beer display closer to its diapers display, having confused correlation and causation.

Purchasing diapers does not cause one to purchase beer (or *vice-versa*); it could simply be that parents of newborns have little time to visit public houses and bars, and whatever drinking they do will be done at home. Who knows? Whatever the case, rumour has it that the experiment was neither popular nor successful.

Applications Typical uses include:

- finding **related concepts** in text documents – looking for pairs (triplets, etc) of words that represent a joint concept: {San Jose, Sharks}, {Michelle, Obama}, etc.;
- detecting **plagiarism** – looking for specific sentences that appear in multiple documents, or for documents that share specific sentences;
- identifying **biomarkers** – searching for diseases that are frequently associated with a set of biomarkers;
- making predictions and decisions based on association rules (there are pitfalls here);
- altering circumstances or environment to take advantage of these correlations (suspected causal effect);
- using connections to modify the likelihood of certain outcomes (see immediately above);
- imputing missing data,
- text autofill and autocorrect, etc.

Other uses and examples can be found in [53, 7, 20].

Causation and Correlation Association rules can automate **hypothesis discovery**, but one must remain correlation-savvy.[16]

16: Which remains less prevalent among quantitative specialists and data scientists than one might hope, unfortunately, in our experience.

If attributes A and B are shown to be correlated in a dataset, there are four possibilities:

- A and B are correlated entirely by chance in this particular dataset;
- A is a relabeling of B (or *vice-versa*);
- A causes B (or *vice-versa*), or
- some combination of attributes $C_1, \ldots, C_n$ (which may not be available in the dataset) cause both A and B.

Siegel [53] illustrates the confusion that can arise with a number of real-life examples:

- Walmart has found that sales of strawberry Pop-Tarts increase about seven-fold in the days preceding the arrival of a hurricane;
- Xerox employees engaged in front-line service and sales-based positions who use Chrome and Firefox browsers perform better on employment assessment metrics and tend to stay with the company longer, or
- University of Cambridge researchers found that liking "Curly Fries" on Facebook is predictive of high intelligence.

It can be tempting to try to **explain** these results (again, from [53]). Perhaps:

- when faced with a coming disaster, people stock up on comfort or nonperishable foods;
- the fact that an employee takes the time to install another browser shows that they are an informed individual and that they care about their productivity, or
- an intelligent person liked this Facebook page first, and her friends saw it, and liked it too, and since intelligent people have intelligent friends (?), the likes spread among people who are intelligent.

While these explanations *might* very well be the right ones (although probably not in the last case), there is **nothing in the data** that supports them. Association rules discovery **finds** interesting rules, but it does not explain them. **The point cannot be over-emphasized**: correlation does not imply causation.

Analysts and consultants might not have much control over the matter, but they should do whatever is in their power so that the following headlines do not see the light of day:

- "Pop-Tarts" get hurricane victims back on their feet;
- Using Chrome of Firefox improves employee performance, or
- Eating curly fries makes you more intelligent.

Definitions A rule $X \longrightarrow Y$ is a statement of the form "if X (the **premise**) then Y (the **conclusion**)" built from any logical combinations of a dataset attributes.

In practice, a rule **does not need to be true for all observations** in the dataset – there could be instances where the premise is satisfied but the conclusion is not.

In fact, some of the "best" rules are those which are only accurate 10% of the time, as opposed to rules which are only accurate 5% of the time, say. As always, **it depends on the context**. To determine a rule's strength, we compute various rule metrics, such as the:

- **support**, which measures the frequency at which a rule occurs in a dataset – low coverage values indicate rules that rarely occur;
- **confidence**, which measures the reliability of the rule: how often does the conclusion occur in the data given that the premises have occurred – rules with high confidence are "truer", in some sense;
- **interest**, which measures the difference between its confidence and the relative frequency of its conclusion – rules with high absolute interest are ... more interesting than rules with small absolute interest;
- **lift**, which measures the increase in the frequency of the conclusion which can be explained by the premises – in a rule with a high lift (> 1), the conclusion occurs more frequently than it would if it were independent of the premises, and
- **conviction** [56], **all-confidence** [40], **leverage** [44], **collective strength** [4], and many others [54, 22].

In a dataset with N observations, let $\text{Freq}(A) \in \{0, 1, \ldots, N\}$ represent the count of the dataset's observations for which property A holds.

This is all the information that is required to compute a rule's evaluation metrics:

$$\begin{aligned}
\text{Support}(X \rightarrow Y) &= \frac{\text{Freq}(X \cap Y)}{N} \in [0, 1] \\
\text{Confidence}(X \rightarrow Y) &= \frac{\text{Freq}(X \cap Y)}{\text{Freq}(X)} \in [0, 1] \\
\text{Interest}(X \rightarrow Y) &= \text{Confidence}(X \rightarrow Y) - \frac{\text{Freq}(Y)}{N} \in [-1, 1] \\
\text{Lift}(X \rightarrow Y) &= \frac{N^2 \cdot \text{Support}(X \rightarrow Y)}{\text{Freq}(X) \cdot \text{Freq}(Y)} \in (0, N^2) \\
\text{Conviction}(X \rightarrow Y) &= \frac{1 - \text{Freq(Y)}/N}{1 - \text{Confidence}(X \rightarrow Y)} \geq 0
\end{aligned}$$

British Music Dataset A simple example will serve to illustrate these concepts. Consider a (hypothetical) music dataset containing data for $N = 15,356$ British music lovers and a **candidate rule** RM:

> "If an individual is born before 1976 (X), then they own a copy of the Beatles' *Sergeant Peppers' Lonely Hearts Club Band*, in some format (Y)".

Let's assume further that

- $\text{Freq}(X) = 3888$ individuals were born before 1976;
- $\text{Freq}(Y) = 9092$ individuals own a copy of *Sergeant Peppers' Lonely Hearts Club Band*, and
- $\text{Freq}(X \cap Y) = 2720$ individuals were born before 1976 and own a copy of *Sergeant Peppers' Lonely Hearts Club Band*.

We can easily compute the 5 metrics for RM:

$$\begin{aligned}
\text{Support(RM)} &= \frac{2720}{15,356} \approx 18\% \\
\text{Confidence(RM)} &= \frac{2720}{3888} \approx 70\% \\
\text{Interest(RM)} &= \frac{2720}{3888} - \frac{9092}{15,356} \approx 0.11 \\
\text{Lift(RM)} &= \frac{15,356^2 \cdot 0.18}{3888 \cdot 9092} \approx 1.18 \\
\text{Conviction(RM)} &= \frac{1 - 9092/15,356}{1 - 2720/3888} \approx 1.36
\end{aligned}$$

These values are easy to interpret: RM occurs in **18%** of the dataset's instances, and it holds true in **70%** of the instances where the individual was born prior to 1976.

This would seem to make RM a **meaningful rule** about the dataset – being older and owning the album are linked properties. But if being younger and not owning that song are not also linked properties, the statement is actually weaker than it would appear at a first glance.

As it happens, RM's lift is **1.18**, which can be rewritten as

$$1.18 \approx \frac{0.70}{0.59},$$

i.e. 59% of ALL individuals also own the song.[17]

17: Whereas 56% of young individuals own the song.

The ownership rates of the two age categories are different, but perhaps not as significantly as one would deduce using the confidence and support alone, which is reflected by the rule's "low" interest, whose value is **0.11**.

Finally, the rule's conviction is **1.36**, which means that the rule would be incorrect 36% more often if X and Y were completely independent.

All this seems to point to the rule RM being not entirely devoid of meaning, but to what extent, exactly? This is a difficult question to answer.[18]

18: There will be times when an interest of 0.11 in a rule would be considered a smashing success; a lift of 15 would not be considered that significant but a support of 2% would be, and so forth.

It is nearly impossible to provide **hard** and **fast** thresholds: it always depends on the context, and on comparing evaluation metric values for a rule with the values obtained for some other of the dataset's rules. In short, evaluation of a lone rule is **meaningless**.

In general, it is recommended to conduct a **preliminary exploration** of the space of association rules (using domain expertise when appropriate) in order to determine reasonable threshold ranges for the specific situation; candidate rules would then be discarded or retained depending on these metric thresholds.

This requires the ability to "easily" generate potentially meaningful candidate rules.

19.3.2 Generating Rules

Given association rules, it is straightforward to evaluate them using various metrics, as discussed in the previous section.

The real challenge of association rules discovery lies in **generating** a set of candidate rules which are likely to be retained, without wasting time generating rules which are likely to be discarded.

An **itemset** (or instance set) for a dataset is a list of attributes and values. A set of **rules** can be created from the itemset by adding "IF ... THEN" blocks to the instances.

As an example, from the instance set

$$\{\text{membership} = \text{True}, \text{age} = \text{Youth}, \text{purchasing} = \text{Typical}\},$$

we can create the 7 following 3−item rules:

- IF (membership = True AND age = Youth) THEN purchasing = Typical;
- IF (age = Youth AND purchasing = Typical) THEN membership = True;
- IF (purchasing = Typical AND membership = True) THEN age = Youth;
- IF membership = True THEN (age = Youth AND purchasing = Typical);

- IF age = Youth THEN (purchasing = Typical AND membership = True);
- IF purchasing = Typical THEN (membership = True) AND age = Youth);
- IF ∅ THEN (membership = True AND age = Youth AND purchasing = Typical);

the 9 following 2−item rules:

- IF membership = True THEN age = Youth;
- IF age = Youth THEN purchasing = Typical;
- IF purchasing = Typical THEN membership = True;
- IF membership = True THEN purchasing = Typical;
- IF age = Youth THEN membership = True;
- IF purchasing = Typical THEN age = Youth;
- IF ∅ THEN (age = Youth AND purchasing = Typical);
- IF ∅ THEN (purchasing = Typical AND membership = True);
- IF ∅ THEN (membership = True) AND age = Youth);

and the 3 following 1−item rules:

- IF ∅ THEN age = Youth;
- IF ∅ THEN purchasing = Typical,
- IF ∅ THEN membership = True.

In practice, we usually only consider rules with the same number of items as there are members in the itemset: in the example above, for instance, the 2−item rules could be interpreted as emerging from the 3 separate itemsets

$$\{\text{membership} = \text{True}, \text{age} = \text{Youth}\}$$
$$\{\text{age} = \text{Youth}, \text{purchasing} = \text{Typical}\}$$
$$\{\text{purchasing} = \text{Typical}, \text{membership} = \text{True}\}$$

and the 1−item rules as arising from the 3 separate itemsets

$$\{\text{membership} = \text{True}\}, \{\text{age} = \text{Youth}\}, \{\text{purchasing} = \text{Typical}\}.$$

Note that rules of the form $\varnothing \rightarrow X$ (or IF ∅ THEN X) are typically denoted simply by X.

Now, consider an itemset $\mathscr{C}_n$ with n members (that is to say, n attribute/level pairs). In an n−item rule derived from $\mathscr{C}$, each of the n members appears either in the premise or in the conclusion; there are thus 2^n such rules, in principle.

The rule where each member is part of the premise (i.e., the rule without a conclusion) is nonsensical and is not allowed; we can derive exactly $2^n - 1$ n−item rules from $\mathscr{C}_n$. Thus, the **number of rules increases exponentially** when the **number of features increases linearly**.

This combinatorial explosion is a problem – it instantly disqualifies the **brute force** approach (simply listing all possible itemsets in the data and generating all rules from those itemsets) for any dataset with a realistic number of attributes.

How can we then generate a small number of **promising** candidate rules, in general?

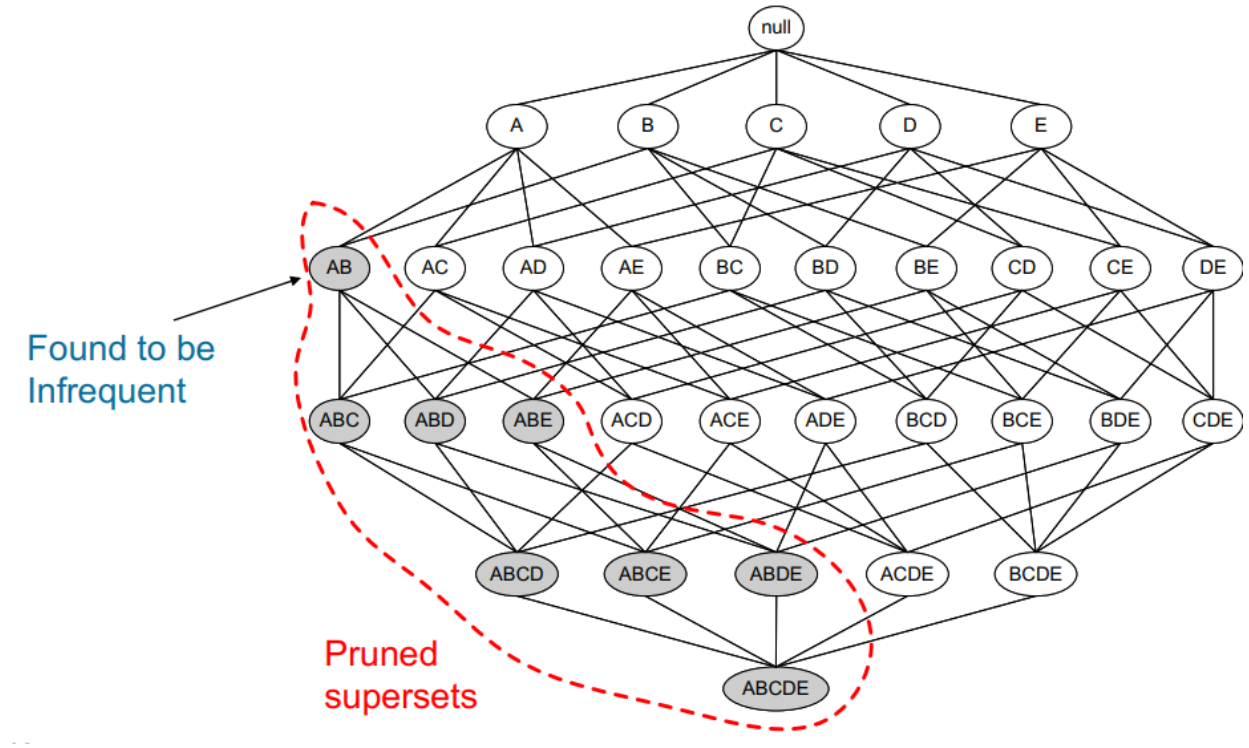


Figure 19.4: Pruned supersets of an infrequent itemset in the *a priori* network of a dataset with 5 items [11]; no rule would be generated from the grey itemsets.

19.3.3 The *A Priori* Algorithm

The ***a priori*** algorithm is an early attempt to overcome that difficulty. Initially, it was developed to work for **transaction data** (i.e. goods as columns, customer purchases as rows), but every reasonable dataset can be transformed into a transaction dataset using dummy variables.

The algorithm attempts to find **frequent itemsets** from which to build candidate rules, instead of building rules from **all** possible itemsets.

It starts by identifying frequent **individual items** in the database and extends those that are retained into larger and larger **item supersets**, who are themselves retained only if they occur **frequently enough** in the data.

The main idea is that "all non-empty subsets of a frequent itemset must also be frequent" [11], or equivalently, that all supersets of an infrequent itemset must also be infrequent (see Figure 19.4).

In the technical jargon of machine learning, we say that *a priori* uses a **bottom-up approach** and the **downward closure property of support**.

The memory savings arise from the fact that the algorithm prunes candidates with **infrequent sub-patterns** and removes them from consideration for any future itemset: if a 1−itemset is not considered to be frequent enough, any 2−itemset containing it is also infrequent (see Figure 19.5 for another illustration).

A list of the 4 teams making the playoffs each year is shown on the left ($N = 20$). Frequent itemsets are generated using the *a priori* algorithms,

NHL Playoff Teams (1942-1967)
{Detroit,Boston,Toronto,Montreal}
{Montreal,Detroit,Toronto,Chicago}
{Montreal,Detroit,Toronto,Boston}
{Montreal,Boston,Chicago,Detroit}
{Montreal,Toronto,Boston,Detroit}
{Detroit,Boston,Montreal,Toronto}
{Detroit,Montreal,Toronto,New York}
{Detroit,Toronto,Montreal,Boston}
{Detroit,Montreal,Toronto,Boston}
{Detroit,Montreal,Boston,Chicago}
{Montreal,Detroit,New York,Toronto}
{Detroit,Montreal,Boston,New York}
{Montreal,New York,Detroit,Boston}
{Montreal,Boston,Chicago,Toronto}
{Montreal,Toronto,Chicago,Detroit}
{Montreal,Toronto,Chicago,New York}
{Toronto,Chicago,Montreal,Detroit}
{Montreal,Chicago,Toronto,Detroit}
{Detroit,Montreal,Chicago,Toronto}
{Chicago,Montreal,Toronto,New York}

1-itemsets	Support
{Boston}	11
{Chicago}	10
{Detroit}	17
{Montreal}	20
{New York}	6
{Toronto}	16

2-itemsets	Support
{Boston,Chicago}	3
{Boston,Detroit}	10
{Boston,Montreal}	11
{Boston, Toronto}	7
{Chicago,Detroit}	7
{Chicago,Montreal}	10
{Chicago,Toronto}	8
{Detroit,Montreal}	17
{Detroit,Toronto}	13
{Montreal,Toronto}	16

3-itemsets	Support
{Boston,Detroit,Montreal}	10
{Detroit,Montreal,Toronto}	13

Rules	Support	Confidence	Lift
IF Boston THEN Detroit	0.50	0.91	1.070
IF Boston AND Montreal THEN Detroit	0.50	0.91	1.070
IF Boston THEN Detroit AND Montreal	0.50	0.91	1.070

Figure 19.5: Association rules for NHL playoff teams (1942-1967): 4 teams (out of 6) made the playoffs each year.

with a support threshold of 10. We see that there are 5 frequent 1−itemsets, in yellow;[19] 6 frequent 2−itemsets are found in the subsequent list of ten 2−itemsets, in green.[20] Only 2 frequent 3−itemsets are found, in orange. Candidate rules are generated from the shaded itemsets; the rules retained by the thresholds

$$\text{Support} \geq 0.5, \text{ Confidence} \geq 0.7, \text{ and Lift} > 1 \text{ (barely)},$$

are shown in the table on the bottom row – the main result is that when Boston made the playoffs, it was not surprising to see Detroit also make the playoffs.[21] Are these rules meaningful at all?

Of course, this process requires a support threshold **input**, for which there is no guaranteed way to pick a "good" value; it has to be set sufficiently high to minimize the number of frequent itemsets that are being considered, but not so high that it removes too many candidates from the **output list**.[22]

The algorithm terminates when no further itemsets extensions are retained, which must occur in datasets with a finite # of categorical levels.

- **Strengths:** easy to implement and to parallelize [36]
- **Limitations:** slow, requires frequent data set scans, not ideal for finding rules for infrequent and rare itemsets

More efficient algorithms have since displaced *a priori* in practice:[23]

- **Max-Miner** tries to identify frequent itemsets without enumerating them – it performs jumps in itemset space instead of using a bottom-up approach;
- **Eclat** is faster and uses depth-first search, but requires extensive memory storage.[24]

19: New York made the playoffs 6 < 10 times – no larger frequent itemset can contain New York.

20: Note the absence of New York.

21: The presence or absence of Montreal in a rule is a red herring, as Montreal made the playoffs every year in the data.

22: As ever, optimal threshold values are **dataset-specific**.

23: Although it retains **historical value**.

24: *A priori* and eclat are both implemented in the `R` package `arules` [40].

19.3.4 Validation

How **reliable** are association rules? What is the likelihood that they occur entirely **by chance**? How **relevant** are they? Can they be generalised **outside** the dataset, or to **new** data streaming in?

These questions are notoriously difficult to answer for association rules discovery, but **statistically sound association discovery** can help reduce the risk of finding spurious associations to a user-specified significance level [54, 22].

We end this section with a few comments.

- Since frequent rules correspond to instances that occur repeatedly in the dataset, algorithms that generate itemsets often try to **maximize coverage**. When **rare events** are more meaningful (such as detection of a rare disease or a threat), we need algorithms that can generate rare itemsets. **This is not a trivial problem**.
- Continuous data has to be binned into **categorical** data to generate rules. As there are many ways to accomplish that task, the same dataset can give rise to completely different rules. This could create some credibility issues with clients and stakeholders.
- Other popular algorithms include: AIS, SETM, aprioriTid, apriori-Hybrid, PCY, Multistage, Multihash, etc.
- Additional evaluation metrics can be found in the `arules` documentation [40].

19.3.5 Case Study: Danish Medical Data

In *temporal disease trajectories condensed from population wide registry data covering 6.2 million patients* [29], A.B. Jensen *et al.* study diagnoses in the Danish population, with the help of association rules mining and clustering methods.

Objectives Estimating **disease progression** (trajectories) from current patient state is a crucial notion in medical studies. Such trajectories had (at the time of publication) only been analyzed for a small number of diseases, or using large-scale approaches without consideration for time exceeding a few years. Using data from the *Danish National Patient Registry* (an extensive, long-term data collection effort by Denmark), the authors sought connections between different **diagnoses**: how does the presence of a diagnosis at some point in time allow for the prediction of another diagnosis at a later point in time?

Methodology The authors took the following methodological steps:

1. compute the **strength of correlation** for pairs of diagnoses over a 5 year interval (on a representative subset of the data);
2. test diagnoses pairs for **directionality** (one diagnosis repeatedly occurring before the other);
3. determine reasonable **diagnosis trajectories** (thoroughfares) by combining smaller (but frequent) trajectories with overlapping diagnoses;

4. **validate** the trajectories by comparison with non-Danish data,
5. **cluster** the thoroughfares to identify a small number of **central medical conditions** (key diagnoses) around which disease progression is organized.

Data The Danish National Patient Registry is an electronic health registry containing administrative information and diagnoses, covering the whole population of Denmark, including private and public hospital visits of all types: inpatient (overnight stay), outpatient (no overnight stay) and emergency. The data set covers 15 years, from January '96 to November '10 and consists of 68 million records for 6.2 million patients.

Challenges and Pitfalls

- Access to the **Patient Registry** is protected and could only be granted after approval by the *Danish Data Registration Agency the National Board of Health*.
- Gender-specific differences in diagnostic trends are clearly identifiable (pregnancy and testicular cancer do not have much cross-appeal), but many diagnoses were found to be made exclusively (or at least, predominantly) in different sites (inpatient, outpatient, emergency ward), which suggests the importance of stratifying by **site** as well as by **gender**.
- In the process of forming small diagnoses chains, it became necessary to compute the correlations using **large groups** for each pair of diagnoses. For close to 1 million diagnosis pairs, more than 80 million samples would have been required to obtain significant $p-$values while compensating for **multiple testing**, which would have translated to a thousand years' worth of computer running time. A pre-filtering step was included to avoid this pitfall.[25]

25: The final trajectories were validated using the full sampling procedure.

Project Summary and Results The dataset was reduced to **1,171 significant trajectories**. These thoroughfares were clustered into patterns centred on 5 key diagnoses central to disease progression:

- **diabetes**;
- **chronic obstructive pulmonary disease** (COPD);
- **cancer**;
- **arthritis**, and
- **cerebrovascular disease**.

Early diagnoses for these central factors can help reduce the risk of adverse outcome linked to future diagnoses of other conditions.

Two author quotes illustrate the importance of these results:

> "The sooner a health risk pattern is identified, the better we can prevent and treat critical diseases." [S. Brunak]

> "Instead of looking at each disease in isolation, you can talk about a complex system with many different interacting factors. By looking at the order in which different diseases

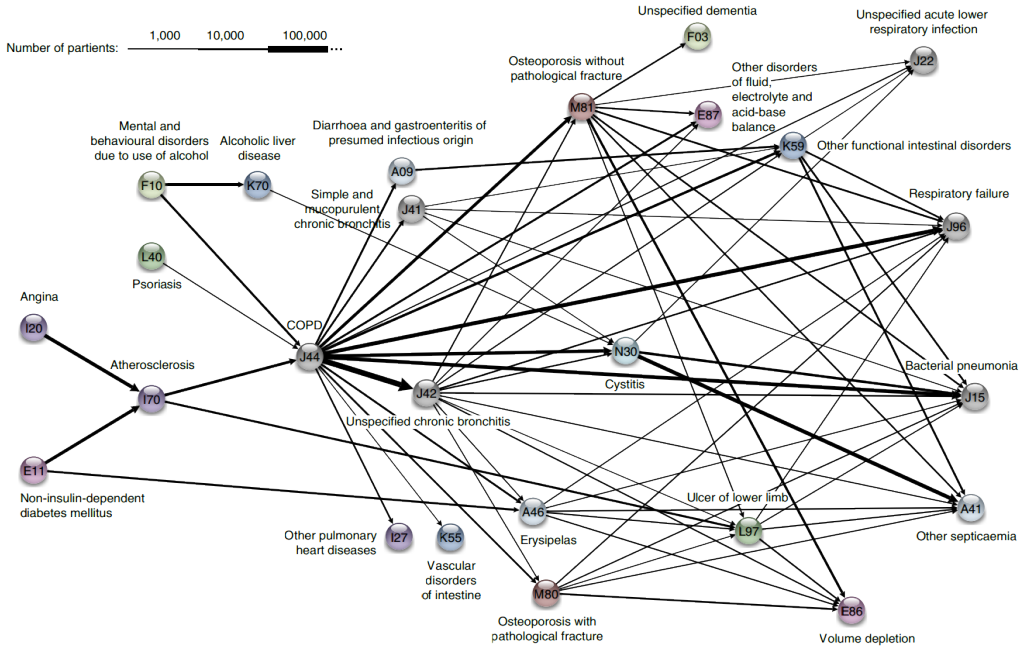


Figure 19.6: The COPD cluster showing five preceding diagnoses leading to COPD and some of its possible outcomes [29].

> appear, you can start to draw patterns and see complex correlations outlining the direction for each individual person."
> [L.J. Jensen]

Among the specific results, the following "surprising" insights were found:

- a diagnosis of anemia is typically followed months later by the discovery of colon cancer;
- gout was identified as a step on the path toward cardiovascular disease, and
- COPD is under-diagnosed and under-treated.

The disease trajectories cluster for COPD is shown in Figure 19.6.

19.3.6 Toy Example: Titanic Dataset

Compiled by Robert Dawson in 1995, the *Titanic* dataset consists of 4 categorical attributes for each of the 2201 people aboard the Titanic when it sank in 1912 (some issues with the dataset have been documented, but we will ignore them for now):

- **class** (1st class, 2nd class, 3rd class, crewmember)
- **age** (adult, child)
- **sex** (male, female)
- **survival** (yes, no)

The natural question of interest for this dataset is:

> "How does survival relate to the other attributes?"

Rule	Supp	Conf	Lift
IF class = 2nd AND age = Child THEN survived = Yes	0.01	1	3.10
IF class = 1st AND sex = Female THEN survived = Yes	0.06	0.97	3.01
IF class = 2nd AND sex = Female THEN survived = Yes	0.04	0.88	2.72
IF class = Crew AND sex = Female THEN survived = Yes	0.00	0.87	2.70
IF class = 2nd AND sex = Male AND age = Adult THEN survived = No	0.07	0.92	1.35
IF class = 2nd AND sex = Male THEN survived = No	0.07	0.86	1.27
IF class = 3rd AND sex = Male AND age = Adult THEN survived = No	0.18	0.84	1.24
IF class = 3rd AND sex = Male THEN survived = No	0.19	0.83	1.22

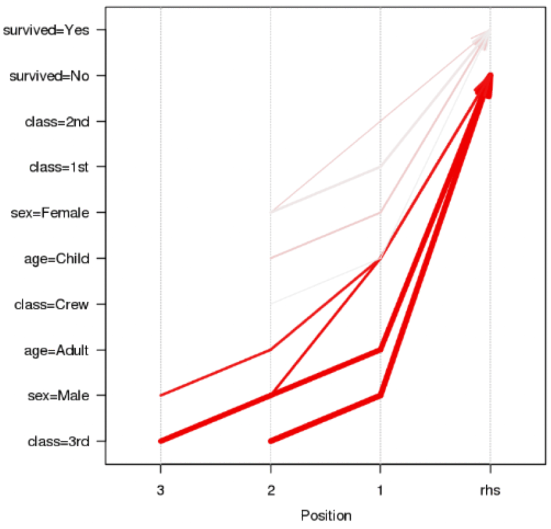


Figure 19.7: Visualization of 8 *Titanic* association rules with parallel coordinates.

This is not, strictly speaking, an unsupervised task (as the interesting rules' structure is fixed to conclusions of the form survival = Yes or survival = No).

For the purpose of this example, we elect not to treat the problem as a **predictive task**, since the long removed situation on the Titanic has little bearing on survival for new data – as such, we use fixed-structure association rules to **describe** and **explore** survival conditions on the *Titanic* (compare with [48]).

We use the `arules` implementation of the *a priori* algorithm in `R` to generate and prune candidate rules, eventually leading to **8 rules** (the results are visualized in Figure 19.7). Who survived? Who didn't?[26] We show how to obtain these rules *via* `R` in Section 19.7.1 (*Association Rules Mining: Titanic Dataset*).

26: Again, with feeling: **correlation does not imply causation.**

19.4 Classification and Value Estimation

> "The diversity of problems that can be addressed by classification algorithms is significant, and covers many domains. It is difficult to comprehensively discuss all the methods in a single book." [2]

19.4.1 Overview

The principles underlying classification, regression and class probability estimation are well-known and straightforward. **Classification** is a supervised learning endeavour in which a sample **training** set of data is used to determine rules and patterns that divide the data into predetermined groups, or classes. The training set usually consists of a **randomly** selected subset of the **labeled** data.[27]

27: **Value estimation** (regression) is similar to classification, except that the target variable is numerical instead of categorical.

In the **testing phase**, the model is used to assign a class to observations in the **testing set**, in which the label is hidden, in spite of being actually known.

The performance of the predictive model is then **evaluated** by comparing the predicted and the actual values for the testing set observations (but **never** using the training set observations). A number of technical issues need to be addressed in order to achieve optimal performance, among them: determining which features to select for inclusion in the model and, perhaps more importantly, which algorithm to choose.

The **edible**/**poisonous** mushroom model from *Data Science and Machine Learning Tasks* provides a clean example of a **classification model**, albeit one for which no detail regarding the training data and choice of algorithm were made available.

Applications Classification and value estimation models are among the most frequently used of the data science models, and form the backbone of what is also known as **predictive analytics**. There are applications and uses in just about every field of human endeavour, such as:

- **medicine and health science** – predicting which patient is at risk of suffering a second, and this time fatal, heart attack within 30 days based on health factors (blood pressure, age, sinus problems, etc.);
- **social policies** – predicting the likelihood of required assisted housing in old age based on demographic information/survey answers;
- **marketing/business** – predicting which customers are likely to cancel their membership to a gym based on demographics and usage;
- in general, predicting that an object belongs to a particular class, or organizing and grouping instances into categories, or
- enhancing the detection of relevant objects:
  - **avoidance** – "this object is an incoming vehicle";
  - **pursuit** – "this object is leaving the scene of a collision";
  - **degree** – "this object is 90% likely to run in front of the car",
- **economics** – predicting the inflation rate for the coming two years based on a number of economic indicators.

Other examples may be found in [33, 32, 18, 31].

Concrete Examples Some concrete examples may provide a clearer picture of the types of SL problems encountered by analysts.

- A motor insurance company has a fraud investigation department that studies up to 20% of all claims made, yet money is still getting lost on fraudulent claims. To help better direct the investigators, management would like to determine, using past data, if it is possible to predict whether a claim is likely to be fraudulent?[28]

28: And/or whether a customer is likely to commit fraud in the near future? Whether an application for a policy is likely to result in a fraudulent claim? If the amount by which a claim will be reduced if it is fraudulent?

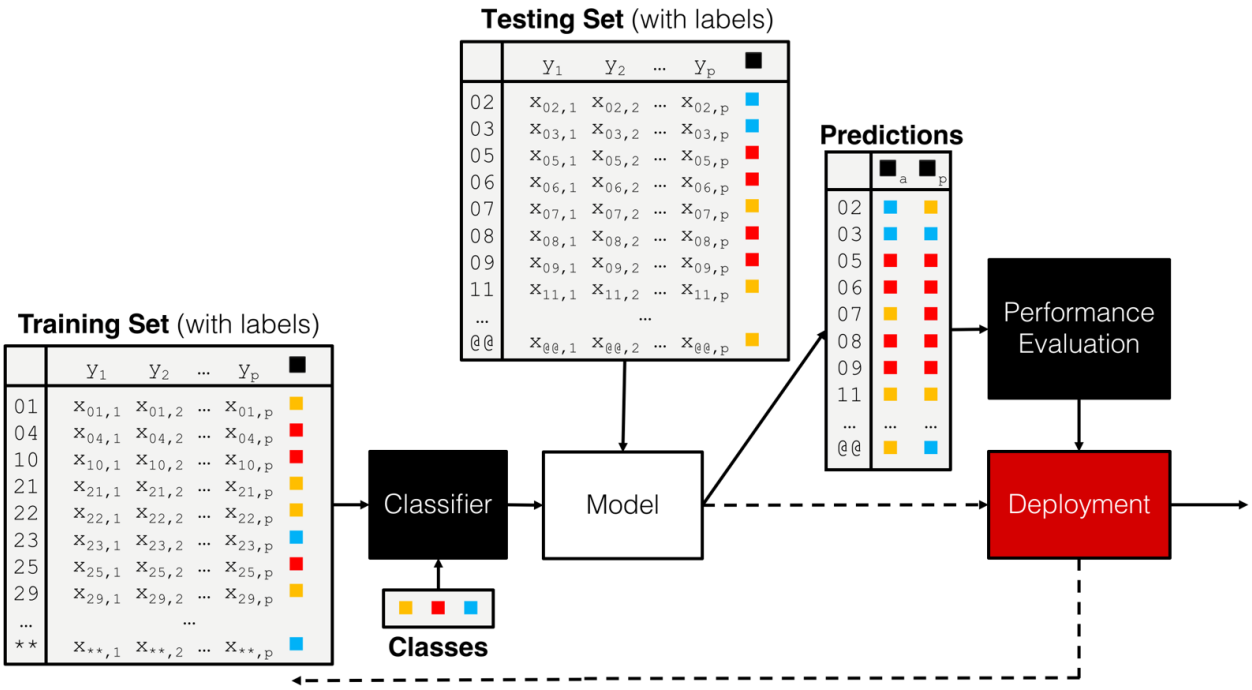


Figure 19.8: A classification pipeline, including training set, testing set, performance evaluation, and (eventual) deployment.

- Customers who make a large number of calls to a mobile phone company's customer service number have been identified as churn risks. The company is interested in reducing said churn. Can they predict the overall lifetime value of a customer? Which customers are more likely to churn in the near future? What retention offer a particular customer will best respond to?

In every classification scenario, the following questions must be answered before embarking on analysis:

- What kind of data is **required**?
- How much of it?
- What would constitute a **predictive success**?
- What are the **risks** associated with a predictive modeling approach?

These have no one-size-fits-all answers; they have to be considered on a case-by-case basis.

In the absence of testing data, classification models cannot be used for predictive tasks, but may still be useful for **descriptive** tasks (see Titanic example above).

When testing data exists, the overall process is often quite similar, independently of the choice of the algorithm (see the classification pipeline shown in Figure 19.8).

This clearly points to the importance of obtaining **good test data**, but keep in mind that this process may be costly and/or difficult to implement, in general. Data scientists often have to enact clever schemes to collect the right "stuff" – consider, for instance, the current procedures used to prove that an online user is not a bot, such as identifying all traffic lights, motorcycles, crosswalks, store fronts, etc. in a picture.[29]

29: It is not in fact the answers that identify a user as human; rather, it is reaction times and mouse movements that betray bots. The answers are used to collect data to train self-driving vehicle AIs.

19.4.2 Classification Algorithms

The number of classification algorithms is truly staggering – it often seems as though new algorithms and variants are put forward on a monthly basis, depending on the task and on the type of data [2].

While some of them tend to be rather esoteric, there is a fairly small number of commonly-used workhorse algorithms/approaches that data scientists and consultants should at least have at their command:[30]

30: Full descriptions: [55, 24, 46].

- **logistic regression** and linear regression are classical models which are often used by statisticians but rarely in a classification setting (the estimated coefficients are often used to determine the features' importance); one of their strengths is that the machinery of standard statistical theory (hypothesis testing, confidence intervals, etc.) is still available to the analyst, but they are easily affected by variance inflation in the presence of predictor multi-collinearity, and the stepwise variable selection process that is typically used is problematic – regularization methods would be better suited in general [25] (see Figure 19.9 for illustrations);
- **neural networks** have become popular recently due to the advent of deep learning; they might provide the prototypical example of a **black box** algorithm as they are hard to interpret; another issue is that they require a fair amount of data to train properly – we will have more to say on the topic in a later chapter;
- **decision trees** are perhaps the most common of all data science algorithms, but they tend to overfit the data when they are not pruned correctly, a process which often has to be done manually (see Figure 19.9 for an illustration) – we shall discuss the pros an cons of decision trees in general in *Decision Trees*;
- **naïve Bayes classifiers** have known quite a lot of success in text mining applications (more specifically as the basis of powerful spam filters), but, embarrassingly, no one is quite sure why they should work as well as they do given that one of their required assumptions (independence of priors) is rarely met in practice (see Figure 19.9 for an illustration);
- **support vector machines** attempt to separate the dataset by "fitting" as wide of a "tube" as possible through the classes (subjected to a number of penalty constraints); they have also known successes, most notably in the field of digital handwriting recognition, but their decision boundaries (the tubes in question) tend to be non-linear and quite difficult to interpret; nevertheless, they may help mitigate some of the difficulties associated with big data (see Figure 19.10 for an illustration);
- **nearest neighbours classifiers** (kNN) basically implement a voting procedure and require very little assumptions about the data, but they are not very stable as adding training points may substantially modify the boundary (see Figures 19.9 and 19.10 for illustrations),

Boosting methods [35, 27] and **Bayesian methods** [49, 16] (also see Chapter 25) are becoming increasingly more popular.

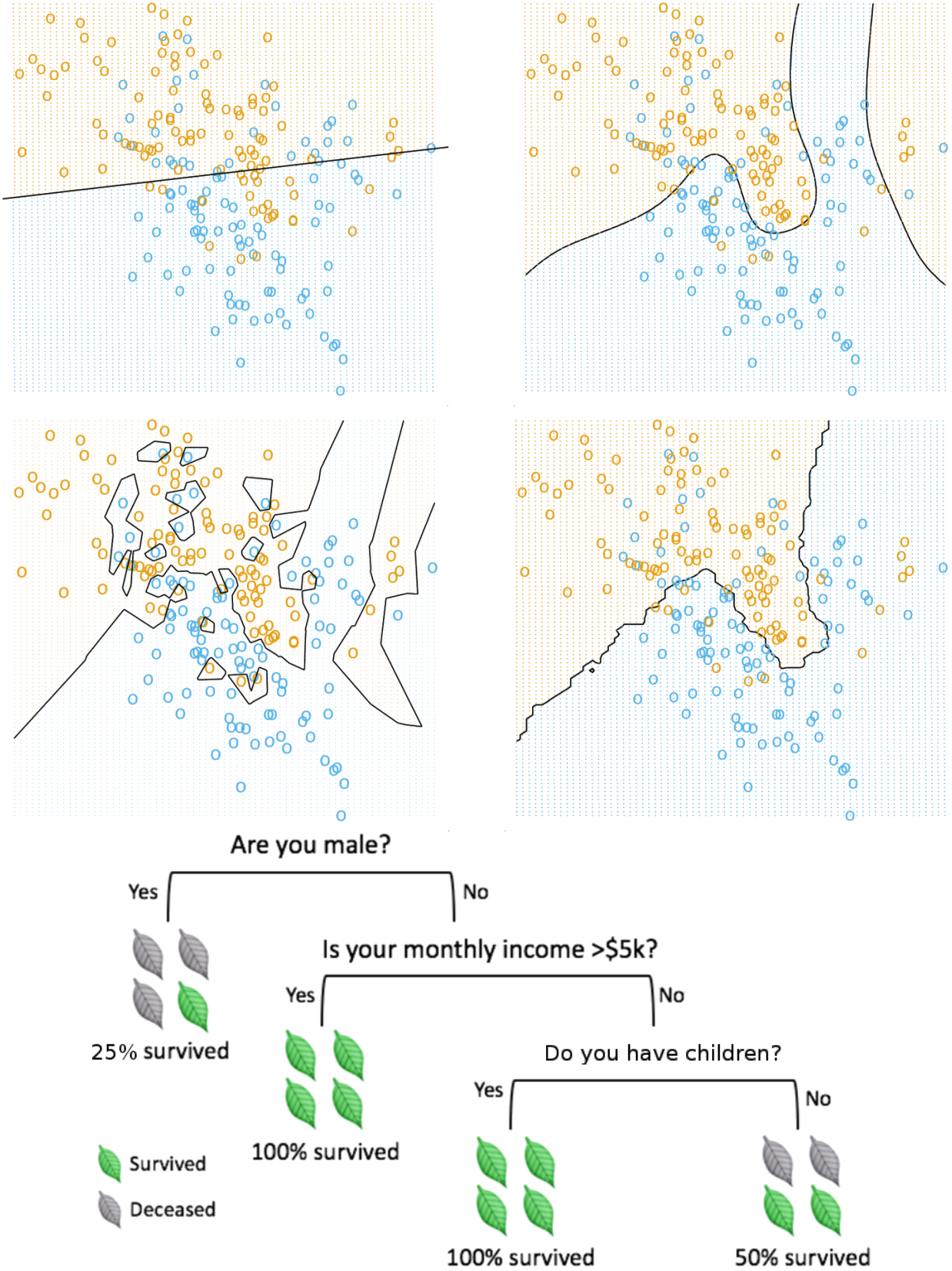


Figure 19.9: Illustrations of various classifiers – linear regression, top left; optimal Bayes, top right; 1NN and 15NN, middle left and right, respectively, on an artificial dataset (from [24]); decision tree depicting the chances of survival for various disasters (fictional, based on [39]). Note that linear regression is more stable, simpler to describe, but less accurate than kNN and optimal Bayes.

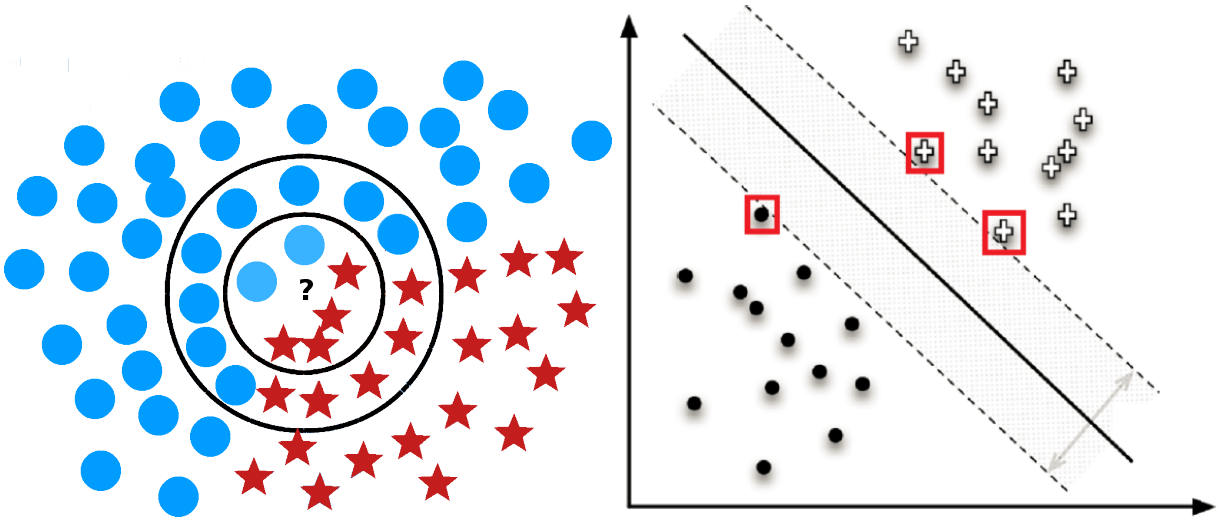


Figure 19.10: Illustration of a k nearest neighbour (left) and a support vector machines classifier (right, based on [46]). What is the 6NN prediction for the location marked by a question mark? What about the 19NN prediction?

19.4.3 Decision Trees

In order to highlight the relative simplicity of most classification algorithms, we will discuss the workings of **ID3**, a historically significant decision tree algorithm.[31]

31: ID3 would never be used in a deployment setting, but it will serve to illustrate a number of classification concepts.

Classification trees are perhaps the most **intuitive** of all supervised methods: classification is achieved by following a path up the tree, from its **root**, through its **branches**, and ending at its **leaves** (although typically the tree is depicted with its root at the top and its leaves at the bottom).

To make a **prediction** for a new instance, it suffices to follow the path down the tree, reading the prediction directly once a leaf is reached. It sounds simple enough in theory, but in practice, creating the tree and traversing it might be **time-consuming** if there are too many variables in the dataset (due to the criterion that is used to determine how the branches split).

Prediction accuracy can be a concern in trees whose growth is **unchecked**. In practice, the criterion of **purity** at the leaf-level[32] is linked to bad prediction rates for new instances. Other criteria are often used to prune trees, which may lead to impure leaves.

32: That is to say, all instances in a leaf belong to the same leaf.

How do we grow such trees?

For predictive purposes, we need a training set and a testing set upon which to evaluate the tree's performance. Ross Quinlan's **Iterative Dichotomizer 3** (a precursor to the widely-used C4.5 and C5.0) follows a simple procedure:

1. split the training data (**parent**) set into (**children**) subsets, using the different levels of a particular attribute;
2. compute the **information gain** for each subset;
3. select the **most advantageous** split, and
4. repeat for each node until some **leaf criterion** is met.

Entropy is a measure of disorder in a set S. Let p_i be the proportion of observations in S belonging to category i, for $i = 1, \ldots, n$. The entropy of S is given by

$$E(S) = -\sum_{i=1}^{n} p_i \log p_i.$$

If the **parent set** S consisting of m records is split into k children sets $C_1, \ldots, C_k$ containing $q_1, \ldots, q_k$ records, respectively, then the **information gained** from the split is

$$I(S : C_1, \ldots, C_k) = E(S) - \frac{1}{m} \sum_{j=1}^{k} q_j E(C_j).$$

The sum term in the information gain equation is a weighted average of the entropy of the children sets.

If the split leads to little disorder in the children, then $\mathrm{IG}(S; C_1, \ldots, C_k)$ is high; if the split leads to similar disorder in both children and parent, then $\mathrm{IG}(S; C_1, \ldots, C_k)$ is low.

Consider, as in Figure 19.11, two splits shown for a parent set with 30 observations separated into 2 classes: ∘ and ★.

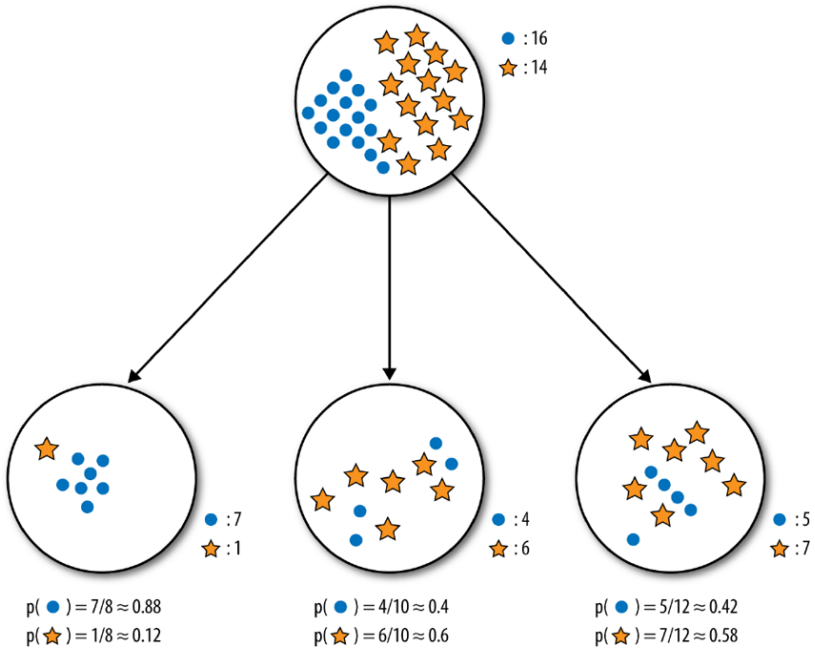


Figure 19.11: Picking the optimal information gain split. [46]

Visually, it appears as though the binary split does a better job of separating the classes. Numerically, the entropy of the parent set S is

$$\begin{aligned} E(S) &= -p_\circ \log p_\circ - p_\star \log p_\star \\ &= -\frac{16}{30} \log \frac{16}{30} - \frac{14}{30} \log \frac{14}{30} \approx 0.99. \end{aligned}$$

For the binary split (on the left), leading to the children set L (left) and R (right), the respective entropies are

$$E(L) = -\frac{12}{13} \log \frac{12}{13} - \frac{1}{13} \log \frac{1}{13} \approx 0.39$$

and

$$E(R) = -\frac{4}{17}\log\frac{4}{17} - \frac{13}{17}\log\frac{13}{17} \approx 0.79,$$

so that the information gained by that split is

$$\mathrm{IG}(S; C_L, C_R) \approx 0.99 - \frac{1}{30}\left[13 \cdot 0.39 + 17 \cdot 0.79\right] = 0.37.$$

On its own, this value is not substantially meaningful – it is only in comparison to the information gained from other splits that it becomes useful. A similar computation for the three-way split leads to $\mathrm{IG}(S; C_1, C_2, C_3) \approx 0.13$, which is indeed smaller than the information gained by the binary split – of these two options, ID3 would select the first as being **most advantageous**.

Decision trees have numerous strengths: they

- are easy to interpret, providing, as they do, a **white box model** – predictions can always be explained by following the appropriate paths;
- can handle numerical and categorical data **simultaneously**, without first having to "binarise" the data;
- can be used with **incomplete** datasets, if needed (although there is still some value in imputing missing observations);
- allow for **built-in feature selection** as less relevant features do not tend to be used as splitting features;
- make **no assumption** about independence of observations, underlying distributions, multi-collinearity, etc., and can thus be used without the need to verify assumptions;
- lend themselves to **statistical validation** (in the form of cross-validation), and
- are in line with **human decision-making approaches**, especially when such decisions are taken deliberately.

On the other hand, they are

- **not usually as accurate** as other more complex algorithms, nor **as robust**, as small changes in the training data can lead to a completely different tree, with a completely different set of predictions;[This can become problematic when presenting the results to a client whose understanding of these matters is slight.]
- particularly **vulnerable to overfitting** in the absence of pruning — and pruning procedures are typically fairly convoluted (some algorithms automate this process, using statistical tests to determine when a tree's "full" growth has been achieved), and
- **biased** towards categorical features with high number of levels, which may give such variables undue importance in the classification process.

Information gain tends to grow small trees in its pursuit of pure leaves, but it is not the only splitting metric in use (**Gini impurity**, **variance reduction**, etc.).

Notes ID3 is a precursor of C4.5, perhaps the most popular decision tree algorithm on the market. There are other tree algorithms, such as C5.0, CHAID, MARS, conditional inference trees, CART, etc., each grown using algorithms with their own strengths and weaknesses.

Regression trees are grown in a similar fashion, but with a **numerical** response variable (predicted inflation rate, say), which introduces some complications [27, 24].

Decision trees can also be combined together using **boosting algorithms** (such as **AdaBoost**) or **random forests**, providing a type of voting procedure also known as **ensemble learning** – an individual tree might make middling predictions, but a large number of judiciously selected trees are likely to make good predictions, on average [27, 24, 35] – we will re-visit these concepts in a later chapter.

Additionally:

- since classification is linked to **probability estimation**, approaches that extend the basic ideas of regression models could prove fruitful;
- **rare occurrences** are often more interesting and more difficult to predict and identify than regular instances – historical data at Fukushima's nuclear reactor prior to the 2011 meltdown could not have been used to learn about meltdowns, for obvious reasons,[33]
- with big datasets, algorithms must also consider **efficiency** – thankfully, decision trees are easily parallelizable.

33: Classical performance evaluation metrics can easily be fooled; if out of two classes one of the instances is only represented in 0.01% of the instances, predicting the non-rare class will yield correct predictions roughly 99.99% of the time, missing the point of the exercise altogether.

19.4.4 Performance Evaluation

As a consequence of the (so-called) **No Free-Lunch Theorem**, no single classifier can be the best performer for every problem [58, 59]. Model selection must take into account:

- the **nature** of the available data;
- the **relative frequencies of the classification sub-groups**;
- the **stated classification goals**;
- how easily the model lends itself to **interpretation** and **statistical analysis**;
- how much **data preparation** is required;
- whether it can accommodate various data types and missing observations;
- whether it performs well with large datasets, and
- whether it is **robust** against small data departures from theoretical assumptions.

Past success is not a guarantee of future success – it is the analyst's responsibility to try a variety of models. But how can the "best" model be selected?

When a classifier attempts to determine what kind of music a new customer would prefer, there is next to no cost in making a mistake; if, on the other hand, the classifier attempts to determine the presence or absence of cancerous cells in lung tissue, mistakes are more consequential. Several metrics can be used to assess a classifier's performance, depending on the context.

Binary classifiers (presented in the abstract in Table 19.1) are simpler and have been studied far longer than multi-level classifiers; consequently, a larger body of evaluation metrics is available for these classifiers.

In the medical literature, for instance, TP, TN, FP and FN stand for **True Positives**, **True Negatives**, **False Positives**, and **False Negatives**, respectively.

		Predicted		
		Category I	Category II	Total
Actuals	Category I	TP	FN	AP
	Category II	FP	TN	AN
	Total	PP	PN	T

Table 19.1: A general binary classifier.

A perfect classifier would be one for which both $\mathrm{FP}, \mathrm{FN} = 0$, but in practice, that rarely ever happens (if at all).

Traditional **performance metrics** include:

- sensitivity: TP/AP
- specificity: TN/AN
- precision: TP/PP
- negative predictive value: TN/PN
- false positive rate: FP/AN
- false discovery rate: $1 - \mathrm{TP}/\mathrm{PP}$
- false negative rate: FN/AP
- accuracy: (TP + TN)/T
- F_1-score: 2TP/(2TP + FP + FN)
- MCC: $\frac{\mathrm{TP}\cdot\mathrm{TN}-\mathrm{FP}\cdot\mathrm{FN}}{\sqrt{\mathrm{AP}\cdot\mathrm{AN}\cdot\mathrm{PP}\cdot\mathrm{PN}}}$
- informedness/ROC: $\mathrm{TP}/\mathrm{AP} + \mathrm{TN}/\mathrm{AN} - 1$
- markedness: $\mathrm{TP}/\mathrm{PP} + \mathrm{TN}/\mathrm{PN} - 1$.

The **confusion matrices** of two artificial binary classifers for a testing set are shown in Table 19.2.

Both classifiers have an accuracy of 80%, but while the second classifier sometimes makes a wrong prediction for A, it never does so for B, whereas the first classifier makes erroneous predictions for both A and B. On the other hand, the second classifier mistakenly predicts occurrence A as B 16 times while the first one only does so 6 times. Which is best?

The performance metrics alone do not suffice to answer the question: the cost associated with making a mistake must also be factored in. Furthermore, it could be preferable to select performance evaluation metrics that generalize more readily to **multi-level classifiers** (see Table 19.3 for examples of associated confusion matrices).

The **accuracy** is the proportion of correct predictions amid all the observations; its value ranges from 0% to 100%. The higher the accuracy, the better the match, and yet, a predictive model with high accuracy may nevertheless be useless thanks to the **Accuracy Paradox** (see rare occurrence sidenote on page 1146).

The **Matthews Correlation Coefficient** (MCC), on the other hand, is a statistic which is of use even when the classes are of very different sizes. As

Actuals	Predicted A	Predicted B	Total	
A	54	10	64	79.0%
B	6	11	17	21.0%
Total	60	21	81	
	74.1%	25.9%		

Classification Rates		Performance Metrics	
Sensitivity:	0.84	Accuracy:	0.80
Specificity:	0.65	F1-Score:	0.87
Precision:	0.90	Informedness (ROC):	0.49
Negative Predictive Value:	0.52	Markedness:	0.42
False Positive Rate:	0.35	M.C.C.:	0.46
False Discovery Rate:	0.10	Pearson's chi2:	0.01
False Negative Rate:	0.16	Hist. Stat:	0.10

Actuals	Predicted A	Predicted B	Total	
A	54	0	54	66.7%
B	16	11	27	33.3%
Total	70	11	81	
	86.4%	13.6%		

Classification Rates		Performance Metrics	
Sensitivity:	1.00	Accuracy:	0.80
Specificity:	0.41	F1-Score:	0.87
Precision:	0.77	Informedness (ROC):	0.41
Negative Predictive Value:	1.00	Markedness:	0.77
False Positive Rate:	0.59	M.C.C.:	0.56
False Discovery Rate:	0.23	Pearson's chi2:	0.33
False Negative Rate:	0.00	Hist. Stat:	0.40

Table 19.2: Performance metrics for two (artificial) binary classifiers.

MCC: 21.6%
Accuracy: 57.6%
Pearson: 0.00079
Hist: 2.8%

Actuals	Predicted Negative	Predicted Postive/Suspected	Predicted Unknown	Total	
Negative	4,156	2,895	362	7,413	45.3%
Postive/Suspected	2,682	5,205	328	8,214	50.2%
Unknown	347	330	68	745	4.6%
Total	7,185	8,429	758	16,372	
	43.9%	51.5%	4.6%		

MCC: 69.7%
Accuracy: 78.3%
Pearson: 0.13161
Hist: 30.0%

Actuals		Predicted Maltreatment: Unfounded	Suspected	Substantiated	Risk: No	Yes	Unknown	Total	
Maltreatment	Unfounded	4,577	-	-	198	6	-	4,781	29.2%
	Suspected	-	965	-	29	2	-	995	6.1%
	Substantiated	-	-	6,187	116	35	2	6,339	38.7%
Risk	No	894	-	763	949	19	9	2,632	16.1%
	Yes	123	-	520	122	111	5	880	5.4%
	Unknown	212	-	303	184	21	24	745	4.6%
Total		5,805	965	7,772	1,597	194	40	16,372	
		35.5%	5.9%	47.5%	9.8%	1.2%	0.2%		

Table 19.3: Performance metrics for (artificial) multi-level classifiers: ternary - left; senary - right [personal files].

a correlation coefficient between the actual and predicted classifications, its range varies from -1 to 1.

If MCC $= 1$, the predicted and actual responses are identical, while if MCC $= 0$, the classifier performs no better than a random prediction ("flip of a coin").

It is also possible to introduce two non-traditional performance metrics (which are nevertheless well-known statistical quantities) to describe how accurately a classifier preserves the classification distribution (rather than how it behaves on an observation-by-observation basis):

- Pearson's χ^2: $\frac{1}{\mathrm{T}}\left((\mathrm{PP}-\mathrm{AP})^2/\mathrm{PP}+(\mathrm{PN}-\mathrm{AN})^2/\mathrm{PN}\right)$
- Hist: $\frac{1}{\mathrm{T}}\left(|\mathrm{PP}-\mathrm{AP}|+|\mathrm{PN}-\mathrm{AN}|\right)$

Note, however, that these are non-standard performance metrics. For a given number of levels, the smaller these quantities, the more similar the actual and predicted distributions.

For **numerical targets** y with predictions $\hat{y}$, the confusion matrix is not defined, but a number of classical performance evaluation metrics can

be used on the testing set: the

- **mean squared** and **mean absolute errors**

$$\text{MSE} = \text{mean}\left\{(y_i - \hat{y}_i)^2\right\}, \quad \text{MAE} = \text{mean}\{|y_i - \hat{y}_i|\};$$

- **normalized mean squared/mean absolute errors**

$$\text{NMSE} = \frac{\text{mean}\left\{(y_i - \hat{y}_i)^2\right\}}{\text{mean}\left\{(y_i - \overline{y})^2\right\}},$$
$$\text{NMAE} = \frac{\text{mean}\left\{|y_i - \hat{y}_i|\right\}}{\text{mean}\left\{|y_i - \overline{y}|\right\}};$$

- **mean average percentage error**

$$\text{MAPE} = \text{mean}\left\{\frac{|y_i - \hat{y}_i|}{y_i}\right\};$$

- **correlation** $\rho_{y,\hat{y}}$, which is based on the notion that for good models, the predicted values and the actual values should congregate around the lines $y = \hat{y}$ (as in Figure 19.12).

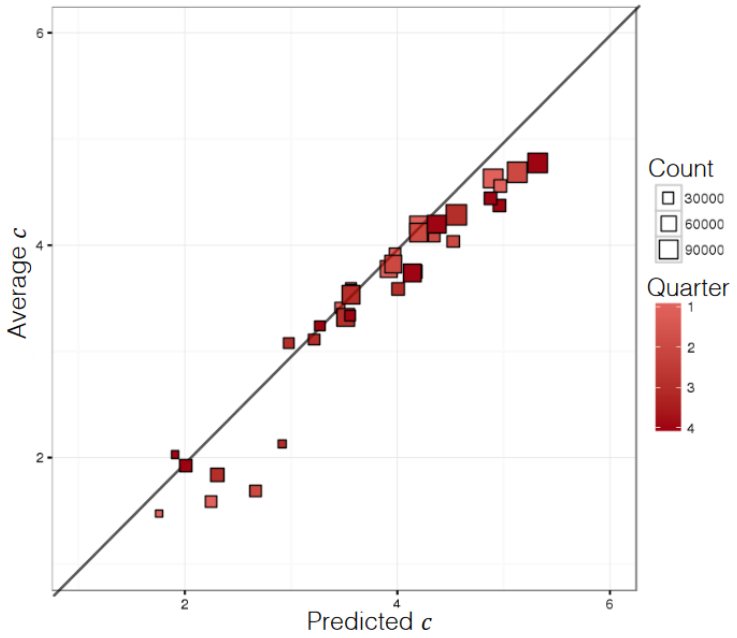


Figure 19.12: Predicted and actual numerical responses [personal file].

As is the case for classification, an isolated value estimation performance metric does not provide enough of a rationale for model validation/selection. One possible exception: **normalized evaluation metrics** do provide some information about the relative quality of performance [24, 27].

19.4.5 Case Study: Minnesota Tax Audit

Large gaps between revenue owed (in theory) and revenue collected (in practice) are problematic for governments. Revenue agencies implement various fraud detection strategies (such as audit reviews) to bridge that gap.

Since business audits are rather costly, there is a definite need for algorithms that can predict whether an audit is likely to be successful or a waste of resources.

In *Data Mining Based Tax Audit Selection: A Case Study of a Pilot Project at the Minnesota Department of Revenue* [26], Hsu et al. study the Minnesota Department of Revenue's (DOR) tax audit selection process with the help of classification algorithms.

Objective The U.S. Internal Revenue Service (IRS) estimated that there were large gaps between **revenue owed** and **revenue collected** for 2001 and for 2006. Using DOR data, the authors sought to increase **efficiency** in the audit selection process and to **reduce the gap** between revenue owed and revenue collected.

Methodology The authors took the following steps:

1. **data selection and separation:** experts selected several hundred cases to audit and divided them into training, testing and validating sets;
2. **classification modeling** using MultiBoosting, Naïve Bayes, C4.5 decision trees, multilayer perceptrons, support vector machines, etc;
3. **evaluation of all models** was achieved by testing the model on the testing set – models originally performed poorly on the testing set until the size of the business being audited was recognized to have an effect, leading to two separate tasks (large and small businesses),
4. **model selection and validation** was done by comparing the estimated accuracy between different classification model predictions and the actual field audits. Ultimately, MultiBoosting with Naïve Bayes was selected as the final model; the combination also suggested some improvements to increase audit efficiency.

Data The data consisted of selected tax audit cases from 2004 to 2007, collected by the audit experts, which were split into training, testing and validation sets:

- the **training data** set consisted of *Audit Plan General* (APGEN) *Use Tax* audits and their results for the years 2004-2006;
- the **testing data** consisted of APGEN Use Tax audits conducted in 2007 and was used to test or evaluate models (for Large and Smaller businesses) built on the training dataset,
- while **validation** was assessed by actually conducting field audits on predictions made by models built on 2007 Use Tax return data processed in 2008.

None of the sets had records in common (see Figure 19.13).

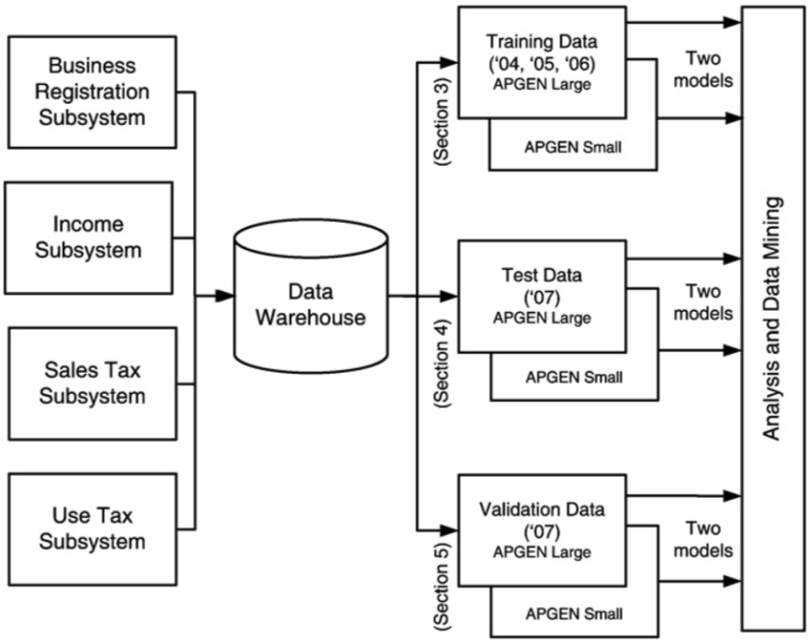


Figure 19.13: Data sources for APGEN mining [26]. Note the 6 final sets which feed the Data Analysis component.

Strengths and Limitations of Algorithms

- The Naïve Bayes classification scheme assumes independence of the features, which rarely occurs in real-world situations. This approach is also known to potentially introduce bias to classification schemes. In spite of this, classification models built using Naïve Bayes have a successful track record.
- MultiBoosting is an **ensemble technique** that uses committee (i.e. groups of classification models) and "group wisdom" to make predictions; unlike other ensemble techniques, it is different from other ensemble techniques in the sense that it forms a committee of sub-committees (i.e., a group of groups of classification models), which has a tendency to reduce both bias and variance of predictions (see [35, 2] for more information on these topics).

Procedures Classification schemes need a response variable for prediction: audits which yielded more than $500 per year in revenues during the audit period were classified as *Good*; the others were *Bad*. The various models were tested and evaluated by comparing the performances of the manual audits (which yield the actual revenue) and the classification models (the predicted classification).

The procedure for manual audit selection in the early stages of the study required:

1. DOR experts selecting several thousand potential cases through a query;
2. DOR experts further selecting several hundreds of these cases to audit;
3. DOR auditors actually auditing the cases, and
4. calculating audit accuracy and return on investment (ROI) using the audits results.

Once the ROIs were available, data mining started in earnest. The steps involved were:

1. **Splitting the data** into training, testing, and validating sets.
2. **Cleaning the training data** by removing "bad" cases.
3. **Building** (and revising) **classification models** on the training dataset. The first iteration of this step introduced a separation of models for larger businesses and relatively smaller businesses according to their **average annual withholding amounts** (the threshold value that was used is not revealed in [26]).
4. **Selecting separate modeling features** for the APGEN Large and Small training sets. The feature selection process is shown in Figure 19.14.
5. **Building classification models** on the training dataset for the two separate class of business (using C4.5, Naïve Bayes, multilayer perceptron, support vector machines, etc.), and assessing the classifiers using **precision** and **recall** with improved estimated ROI:

$$\text{Efficiency} = \text{ROI} = \frac{\text{Total revenue generated}}{\text{Total collection cost}}$$

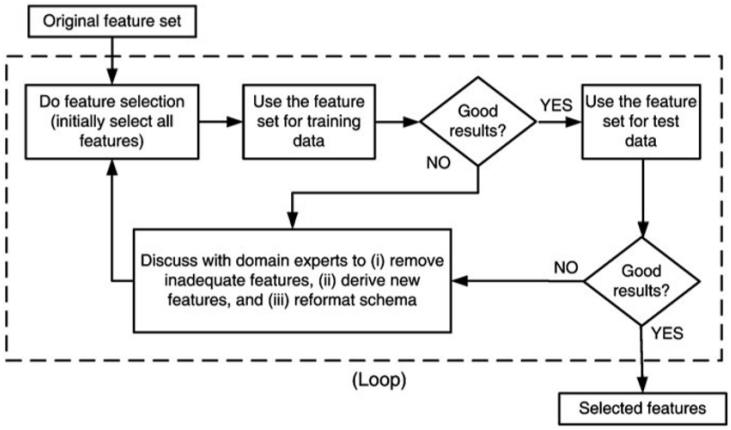


Figure 19.14: Feature selection process in [26]; note the involvement of domain experts.

Results, Evaluation and Validation The models that were eventually selected were combinations of MultiBoosting and Naïve Bayes (C4.5 produced interpretable results, but its performance was shaky). For APGEN Large (2007), experts had put forward 878 cases for audit (495 of which proved successful), while the classification model suggested 534 audits (386 of which proved successful). The theoretical best process would find 495 successful audits in 495 audits performed, while the manual audit selection process needed 878 audits in order to reach the same number of successful audits.

For APGEN Small (2007), 473 cases were recommended for audit by experts (only 99 of which proved successful); in contrast, 47 out of the 140 cases selected by the classification model were successful. The theoretical best process would find 99 successful audits in 99 audits performed, while the manual audit selection process needed 473 audits in order to reach the same number of successful audits.

In both cases, the classification model improves on the manual audit process: roughly 685 data mining audits to reach 495 successful audits of

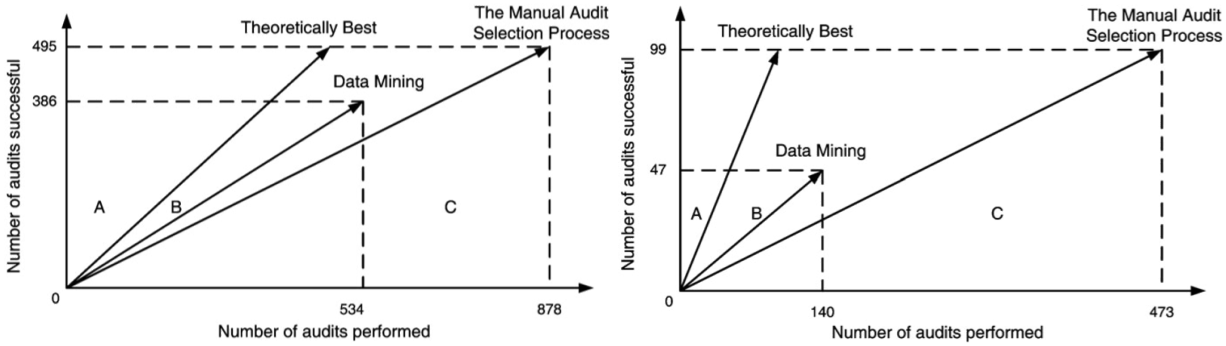


Figure 19.15: Audit resource deployment efficiency [26]; left – APGEN Large (2007); right – APGEN Small (2007). In both cases, the Data Mining approach was more efficient (the slope of the Data Mining vector is "closer" to the Theoretical Best vector than is the Manual Audit vector).

	Predicted as good	Predicted as bad
Actually good	386 (Use tax collected)	109 (Use tax lost)
	R = $5,577,431 (83.6 %)	R = $925,293 (13.9 %)
	C = $177,560 (44 %)	C = $50,140 (12.4 %)
Actually bad	148 (costs wasted)	235 (costs saved)
	R = $72,744 (1.1 %)	R = $98,105 (1.4 %)
	C = $68,080 (16.9 %)	C = $108,100 (26.7 %)

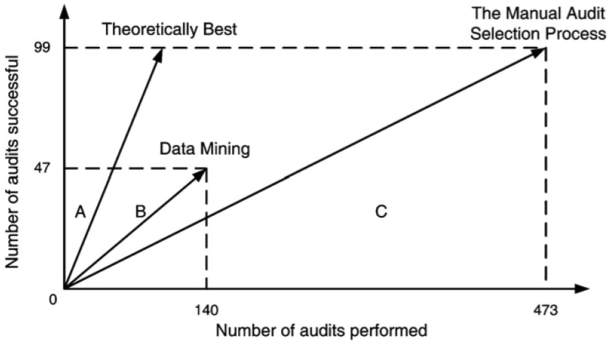


Table 19.4: Confusion matrices for audit evaluation [26]; left – APGEN Large (2007); right – APGEN Small (2007). *R* stands for revenues, *C* for collection costs.

APGEN Large (2007), and 295 would be required to reach 99 successful audits for APGEN Small (2007), as can be seen in Figure 19.15.

Figure 19.4 presents the confusion matrices for the classification model on both the APGEN Large and Small 2007 datasets.

The revenue *R* and collection cost *C* entries can be read as follows: the 47 successful audits which were correctly identified by the model for APGEN Small (2007) correspond to cases consuming 9.9% of collection costs but generating 42.5% of the revenues. Similarly, the 281 bad audits correctly predicted by the model represent notable collection cost savings. These are associated with 59.4% of collection costs but they generate only 11.1% of the revenues.

Once the testing phase of the study was completed, the DOR validated the data mining-based approach by using the models to select cases for actual field audits in a real audit project. The prior success rate of audits for APGEN Use tax data was 39% while the model was predicting a success rate of 56%; the actual field success rate was 51%.

Take-Aways A substantial number of models were churned out before the team made a final selection. Past performance of a model family in a previous project can be used as a guide, but it provides no guarantee regarding its performance on the current data – remember the *No Free Lunch (NFL) Theorem* [58]: nothing works best all the time!

There is a definite iterative feel to this project: the feature selection process could very well require a number of visits to domain experts before the

feature set yields promising results. This is a valuable reminder that the data analysis team should seek out individuals with a good understand of both data and context. Another consequence of the NFL is that domain-specific knowledge has to be integrated in the model in order to beat random classifiers, on average [59].

Finally, this project provides an excellent illustration that even slight improvements over the current approach can find a useful place in an organization – data science is not solely about Big Data and disruption!

19.4.6 Toy Example: Kyphosis Dataset

As a basic illustration of these concepts, consider the following example. **Kyphosis** is a medical condition related to an excessive convex curvature of the spine. Corrective spinal surgery is at times performed on children.

A dataset of 81 observations and 4 attributes has been collected (we have no information on how the data was collected and how representative it is likely to be, but those details can be gathered from [9]).

The attributes are:

- **kyphosis** (absent or present after presentation);
- **age** (at time of operation, in months);
- **number** (of vertebrae involved),
- **start** (topmost vertebra operated on).

The natural question of interest for this dataset is:

> "How do the three explanatory attributes impact the operation's success?"

We use the `rpart` implementation of Classification and Regression Tree (CART) in `R` to generate a decision tree. Strictly speaking, this is not a predictive supervised task as we treat the entire dataset as a training set for the time being – there are no hold-out testing observations.

The results are shown in Figure 19.16. Interestingly, it would appear that the variable `number` does not play a role in determining the success of the operation (for the observations in the dataset).

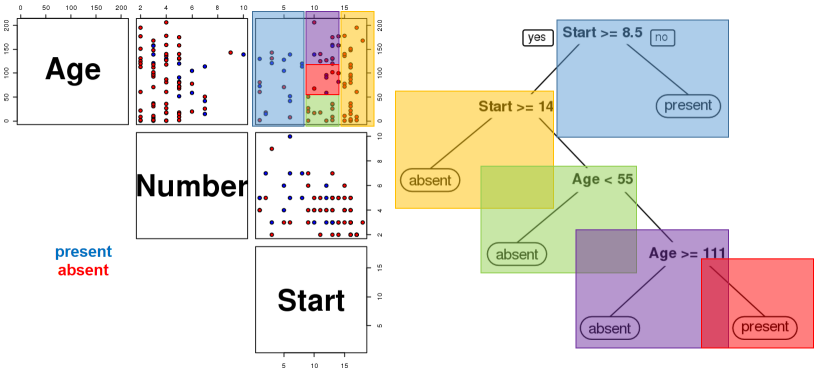


Figure 19.16: Kyphosis decision tree visualization. Only two features are used to construct the tree. We also note that the leaves are not pure – there are blue **and** red instances in 3 of the 5 classification regions.

Furthermore, the decision tree visualization certainly indicates that its leaves are not pure (see Figure 19.17. Some additional work suggests that the tree is somewhat overgrown and that it could benefit from being pruned after the first branching point.

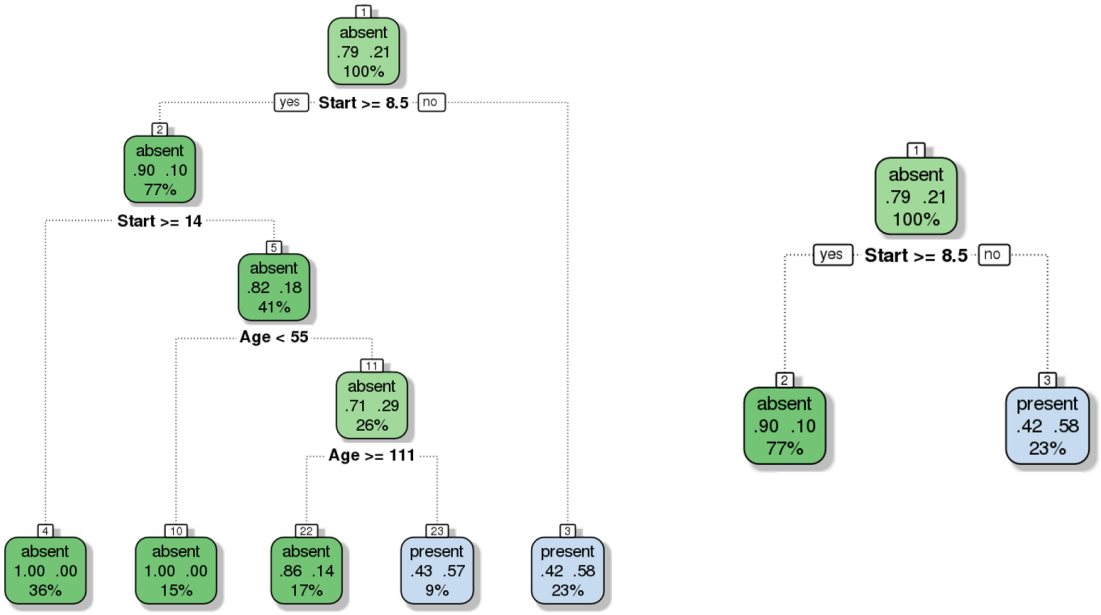


Figure 19.17: Pruning a decision tree – the original tree (left) is more accurate/more complex than the pruned tree (right).

		Predicted A	Predicted B		Total
Actuals	A	23	3	26	83.9%
	B	3	2	5	16.1%
Total		26	5	31	
		83.9%	16.1%		

Classification Rates		*Performance Metrics*	
Sensitivity:	0.88	**Accuracy:**	0.81
Specificity:	0.40	**F1-Score:**	0.88
Precision:	0.88	**Informedness (ROC):**	0.28
Negative Predictive Value:	0.40	**Markedness:**	0.28
False Positive Rate:	0.60	**M.C.C.:**	0.28
False Discovery Rate:	0.12	**Pearson's chi2:**	0.00
False Negative Rate:	0.12	**Hist. Stat:**	0.00

Table 19.5: Kyphosis decision tree – performance evaluation. The accuracy and F_1 scores are good, but the false discovery and false negative rates are not so great. This tree is good at predicting successful surgeries, but not fantastic at predicting failed surgeries. Is it still useful?

At any rate, it remains meangingless to discuss the performance of the tree for **predictive purposes** if we are not using a holdout testing sample (not to say anything about the hope of generalizing to a larger population).

To that end, we trained a model on 50 randomly selected observations and evaluated the performance on the remaining 31 observations (the structure of the tree is not really important at this stage). The results are shown in Table 19.5. Is the model "good"?

It is difficult to answer this question in the machine learning sense without being able to compare its performance metrics with those of other models (or families of models).[34]

In the *Model Selection* subsection, we will briefly discuss how estimate a model's true predictive error rate through **cross-validation**. We will also discuss a number of other issues that can arise when ML/AI methods are not used correctly.

We show how to obtain these decision trees *via* R in Section 19.7 (*Classification: Kyphosis Dataset*).

34: The relative small size of the dataset should give data analysts pause for thought, at the very least.

19.5 Clustering

> "Clustering is in the eye of the beholder, and as such, researchers have proposed many induction principles and models whose corresponding optimisation problem can only be approximately solved by an even larger number of algorithms." [17]

19.5.1 Overview

We can make a variety of **quantitative statements** about a dataset, at the **univariate** level. For instance, we can

- compute **frequency counts** for the variables;
- identify **measures of centrality** (mean, mode, median), and
- measure the **dispersion** (range, standard deviation), among others.

At the **multivariate** level, the various options include n**−way tabulations**, **correlation analysis**, and **data visualization**, among others.

While these can provide insights in simple situations, datasets with a **large number of variables** or with **mixed types** (categorical and numerical) might not yield to such an approach. Instead, insights might come in the form of **aggregation** or **clustering** of similar observations.

A successful **clustering scheme** is one that tightly joins together any number of similarity profiles – "tight" in this context refers to small variability within the cluster, see Figure 19.18 for an illustration.

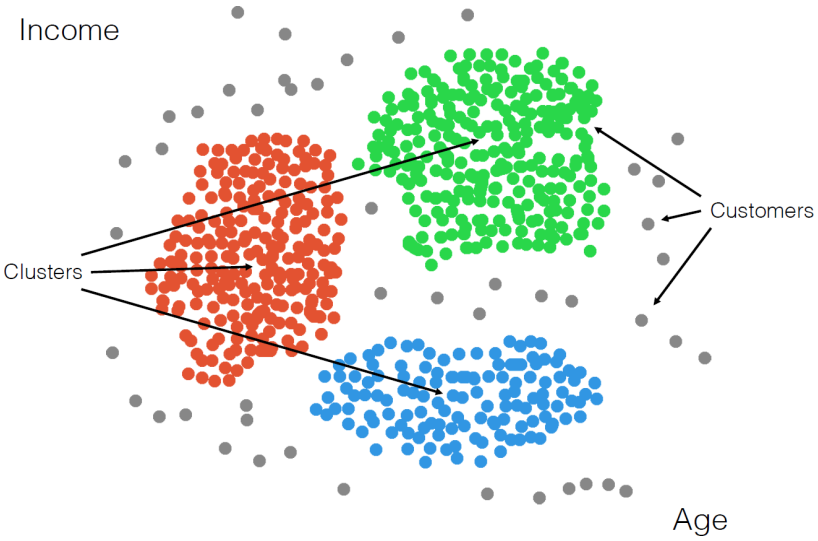


Figure 19.18: Clusters and outliers in an artificial dataset [personal file].

A typical application is one found in **search engines**, where the listed search results are the nearest similar objects (**relevant webpages**) clustered around the search item.

Dissimilar objects (**irrelevant webpages**) should not appear in the list, being "far" from the search item. Left undefined in this example is the crucial notion of **closeness**: what does it mean for one observation to be near another one? Various **metrics** can be used (see Figure 19.19 for some simple examples), and not all of them lead to the same results.

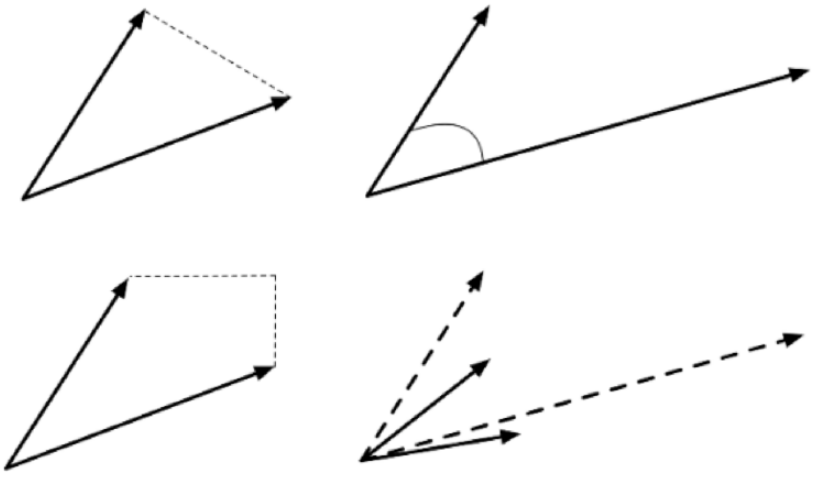


Figure 19.19: Distance metrics between observations: Euclidean (as the crow flies, top left); cosine (direction from a vantage point, top right); Manhattan (taxi-cab, bottom left). Observations should be transformed (scaled, translated) before distance computations (bottom right).

Clustering is a form of **unsupervised learning** since the cluster labels (and possibly their number) are not determined ahead of the analysis.

The algorithms can be **complex** and **non-intuitive**, based on varying notions of similarities between observations, and yet, the temptation to provide a simple *a posteriori* explanation for the groupings remains **strong** – we really, really want to reason with the data.[35]

They are also (typically) **non-deterministic** – the same routine, applied twice to the same dataset, can discover completely different clusters.[36]

This (potential) non-replicability is not just problematic for validation – it can also leads to client dissatisfaction. If the analyst is tasked with finding customer clusters for marketing purposes and the clusters change every time the client or the stakeholders ask for a report, they will be very confused (and will be doubting the results) unless the **stochastic nature** of the process has already been explained.

Another interesting aspect of clustering algorithms is that they often find clusters even when there are no natural ways to break down a dataset into constituent parts.

When there is no natural way to break up the data into clusters, the results may be **arbitrary** and fail to represent any underlying reality of the dataset. On the other hand, it could be that while there was no **recognized** way of naturally breaking up the data into clusters, the algorithm **discovered** such a grouping – clustering is sometimes called **automated classification** as a result.

The aim of clustering, then, is to divide into **naturally occurring groups**. Within each group, observations are similar; between groups, they are dissimilar (see Figure 19.20 for an illustration).

35: Is it possible to look at Figure 19.18 without assigning labels or trying to understand what type of customers were likely to be young and have medium income? Older and wealthier?

36: The order in which the data is presented can play a role, as can starting configurations.

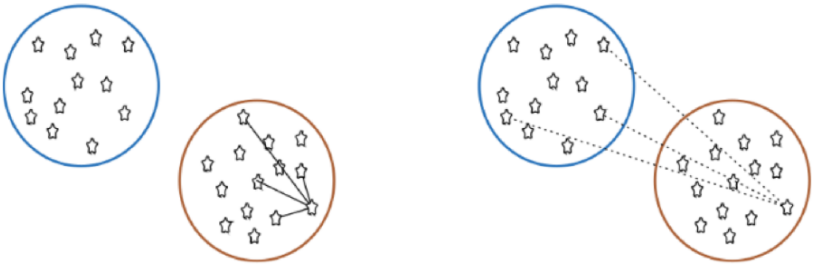


Figure 19.20: Distance to points in own clusters (left, smaller is better) and to points in other clusters (right, larger is better).

As a learning process, clustering is fairly **intuitive** for human beings – our brains unconsciously search for patterns and they can generally handle **messy** data with the same relative ease as clean data. Computers have a harder time of it, however, partly because there is no **agreed-upon definition of what constitutes a cluster**, and so we cannot easily code their recognition into algorithms – to paraphrase Justice Potter Stewart,

> "I may not be able to define what a cluster is, but I know one when I see one."

All clustering algorithms rely on the notion of **similarity** w between observations; in many instances, similarity is obtained *via* a **distance** (or metric) d, with $w \to 1$ as $d \to 0$, and $w \to 0$ as $d \to \infty$. However, there are similarity measures which are not derived from a distance metric.

One additional clustering challenge is that there is no such thing as **the** distance or **the** similarity measure between observations – observations which are similar using a specific measure **may not be similar at all using another**. Commonly-used metrics include:

> euclidean, Manhattan, cosine, Canberra, Hamming, Jaccard, Pearson, and so on.

Note, however, that no matter which similarity measure is selected, the data must first be **transformed**: scaled, centered, etc. (see Figure 19.19). This introduces another layer of arbitrary choices, as there are multiple available options and no **canonical** way to perform this.

Applications Frequently, we use clustering and other unsupervised learning tasks as **preliminary steps** in supervised learning problems, but there exist stand-alone applications as well:

- **text analysis** – grouping similar documents according to their topics, based on the patterns of common and unusual terms;
- **product recommendations** – grouping online purchasers based on the products they have viewed, purchased, liked, or disliked, or grouping products based on customer reviews;
- **marketing** – grouping client profiles based on their demographics and preferences;
- **social network analysis** – recognizing communities within large groups of people;
- **medical imaging** – differentiating between different tissue types in a 3D voxel;
- **genetics** – inferring structures in populations;
- dividing a larger group (or area, or category) into smaller groups, with members of the smaller groups having similarities of some kind, as analytical tasks may then be solved separately for each of the smaller groups, which may lead to increased accuracy once the separate results are aggregated, or
- creating (new) taxonomies on the fly, as new items are added to a group of items, which could allow for easier product navigation on a website like Netflix, for instance.

Numerous other applications may be found in [3, 52, 13, 21, 41, 45, 42, 43, 28, 51, 12, 34, 5].

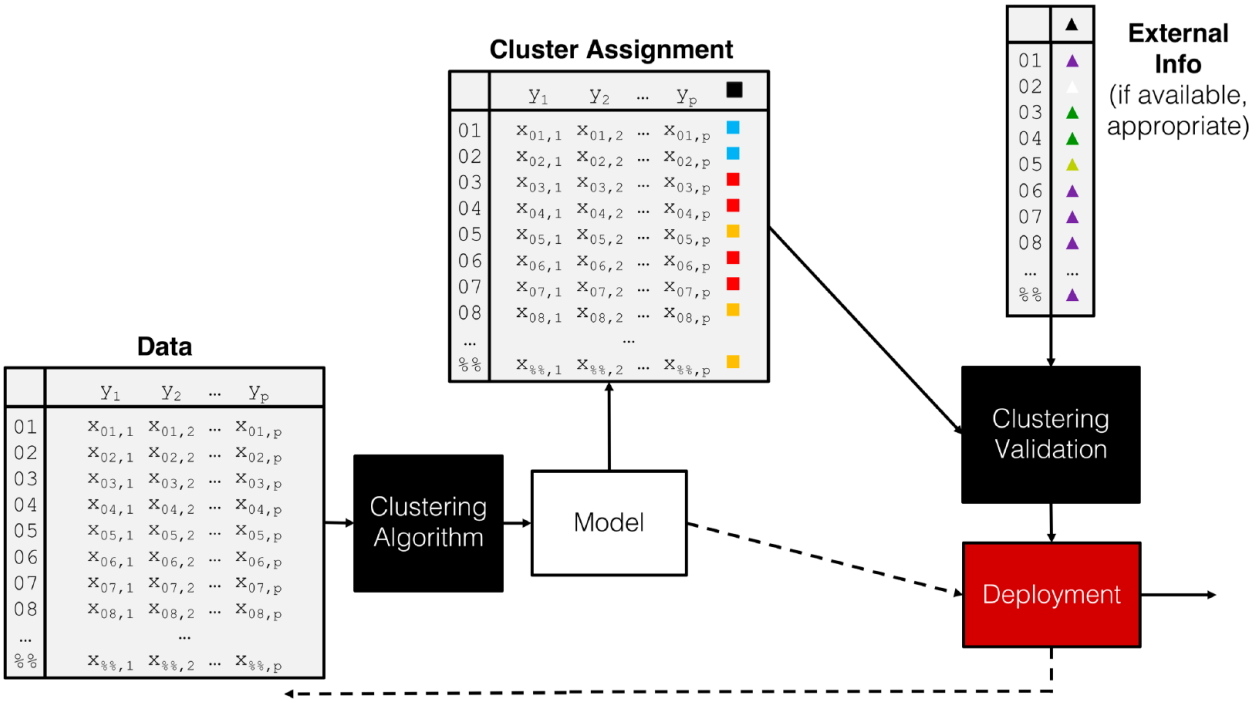


Figure 19.21: A clustering pipeline, including validation and (eventual) deployment.

When all is said and done, the clustering process is quite standard, notwithstanding the choice of scaling strategy, similarity measure, and algorithm and parameters (see the pipeline shown in Figure 19.21).

19.5.2 Clustering Algorithms

As is the case with classification, the number of clustering algorithms is quite high; the Wikipedia page lists 40+ such algorithms as of August 2018 [57]. The choice of algorithms (and associated parameters) is as much an art as it is a science, although domain expertise can come in handy [3].

There is a smaller list of common algorithms that data scientists and consultants should have in their toolbox:[37]

37: Full descriptions: [55, 46, 3].

- $k-$**means**, close on the heels of decision trees for the title of "most-used data science algorithm", is a **partition clustering method** which tends to produce equal-sized clusters; when clients ask for their data to be clustered, they are typically envisioning $k-$means with the Euclidean metric; variants include $k-$mode (for categorical data), $k-$medians (for data with outliers), and $k-$means|| and $k-$means++ for large data sets; the number of clusters k (and the similarity measure/distance metric) must be provided by the user; works fairly well for "blob"-like data;
- **hierarchical clustering** is one of the few deterministic algorithms on the market, with **divisive** (DIANA) and **agglomerative** (AGNES) versions; no parameters need to be inputted, but the users must select a **linkage** strategy (roughly speaking, a metric that computes the distance between clusters) and a level at which to read off the clusters (see Figure 19.22 for an illustration);

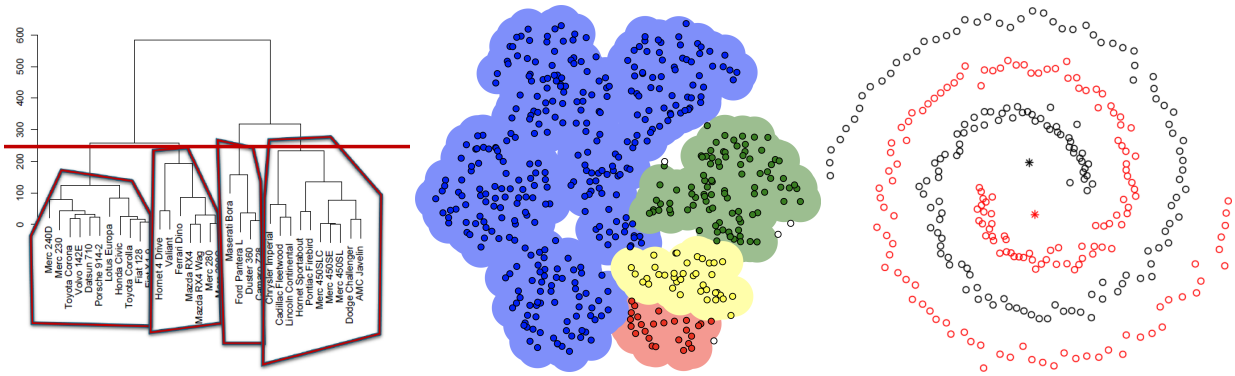


Figure 19.22: Illustration of hierarchical clustering (left), DBSCAN (middle, based on [23]), and spectral clustering (right).

- **density-based spatial clustering** (DBSCAN) is a graph-based approach which attempts to identify densely-packed regions in the dataset; its most obvious advantages (and of its variants OPTICS and DENCLUE) are **robustness to outliers** and not needing to input a **number of clusters** to search for in the data; the main disadvantage is that the optimal input parameters (**neighbourhood radius** and **minimum number of points** to be considered dense) are not easy to derive (see Figure 19.22);
- **affinity propagation** is another algorithm which selects the optimal number of clusters directly from the data, but it does so by trying and evaluating various scenarios, which may end up being **time-consuming**,
- **spectral clustering** can be used to recognize **non-globular** clusters (see Figure 19.22 for an illustration); these are found by computing eigenvalues of an associated Laplacian matrix – consequently, spectral clustering is **fast**.

Other methods include **latent Dirichlet allocation** (used in topics modeling), **expectation-maximisation** (particularly useful to find gaussian clusters), **BIRCH** (a local method which does not require the entire dataset to be scanned) and **fuzzy clustering** (a soft clustering scheme in which the observations have a degree of belonging to each cluster).

19.5.3 k-Means

As mentioned previously, $k-$means is a very natural way to group observations together (formally, $k-$means is linked to **Voronoi tilings**). $k-$means clustering is achieved by:

1. selecting a **distance metric** d (based on the data type and domain expertise);
2. selecting a **number of clusters** k;
3. randomly choosing k data instances as initial **cluster centres**;
4. calculating the **distance** from each observation to each centre;
5. placing each instance in the cluster whose centre it is **nearest** to;
6. computing/updating the **centroid** for each cluster (see Figure 19.23 for an illustration),
7. repeating steps 4-6 until the clusters are **"stable"**.

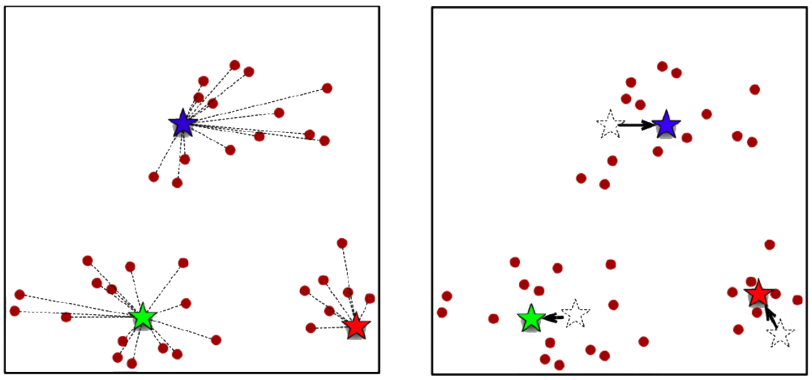


Figure 19.23: $k-$means cluster allocation (left) and updated centres (right) [author unknown].

For $k-$means, cluster centroids are obtained by averaging all points in the cluster. For $k-$medians and $k-$mode, the centrality measure is replaced by the obvious candidate.

This simple algorithm has numerous strengths:

- it is elegant and **easy to implement** (without actually having to compute pairwise distances), and so is extremely common as a result;
- in many contexts, it is a **natural way** to look at grouping observations, and
- it provides a first-pass **basic understanding of the data structure**.

On the other hand,

- it can only assign an instance to **one** cluster, which can lead to overfitting – a more robust solution would be to compute the probability of belonging to each cluster, perhaps based on the distance to the centroid;
- it requires the "true" underlying clusters to be **gaussian-** or blob-shaped, and it will fail to produce useful clusters if that assumption is not met in practice,
- it does not allow for **overlapping** or **hierarchical** groupings.

Notes Let us now return to some issues relating to clustering **in general** (and not just to $k-$means):

No matter the choice of algorithm, clustering rests on the assumption that **nearness of observations** (in whatever metric) is linked with **object similarity**, and that **large distances** are linked with **dissimilarity**. While there are plenty of situations where this is an appropriate assumption to make (temperature readings on a map, for instance), there are others where it is unlikely to be the case (chocolate bars and sensationalist tabloids at a grocery's checkout, say).

The **lack of a clear-cut definition** of what a cluster actually is (see Figure 19.24 for an example) makes it difficult to validate clustering results. Much more can be said on the topic [3].

The fact that various algorithms are **non-deterministic** is also problematic – clustering schemes should never be obtained using **only one** algorithmic pass, as the outcome could be different depending on the **location** of random starting positions and the distance/similarity metric in use.

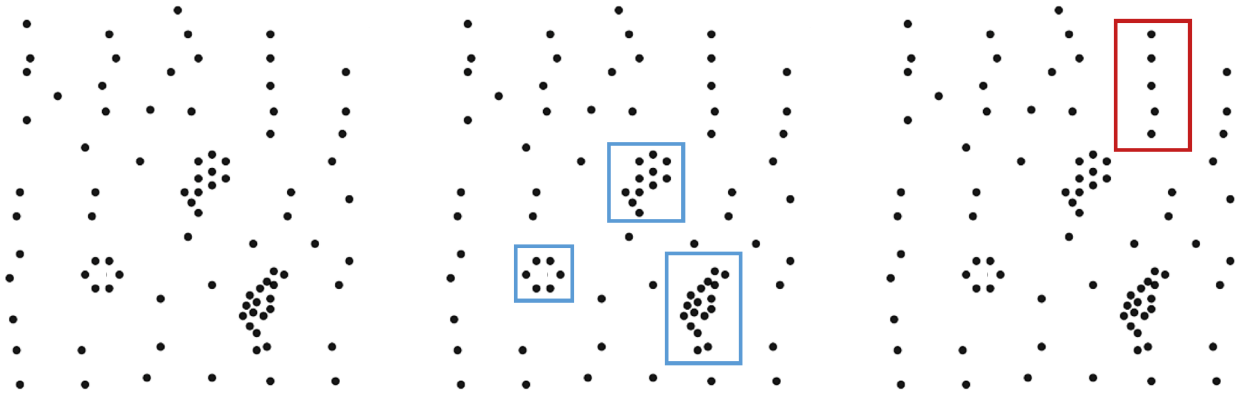


Figure 19.24: Cluster suggestions in an artificial dataset: suggested (blue), rejected (red).

But this apparent fickleness is not necessarily a problem: **essential patterns** may emerge if the algorithms are implemented multiple times, with different starting positions and re-ordered data (see **cluster ensembles** [3]). For those algorithms that require the **number of clusters** as an input, it may be difficult to determine what the optimal number should be (see Figure 19.25 for an illustration).

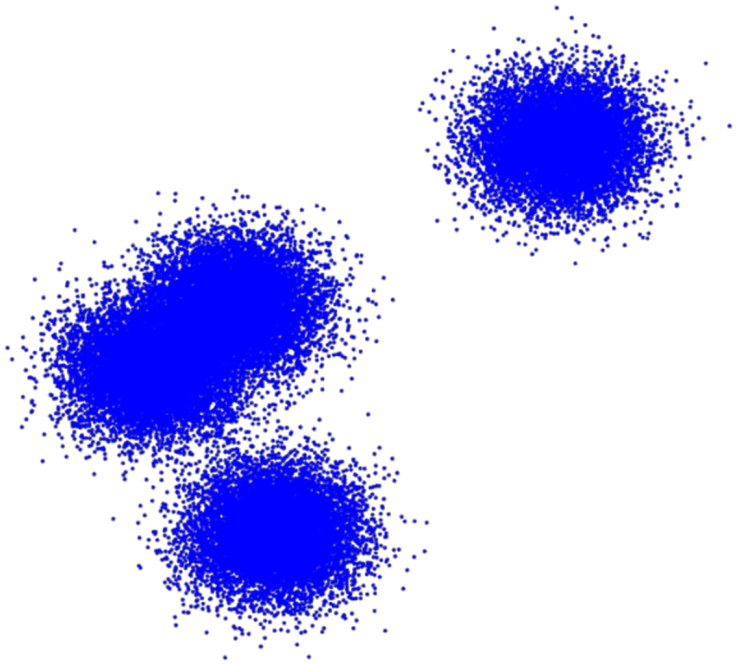


Figure 19.25: The number of clusters in a dataset is ambiguous: are there 2, 3, 4+ clusters in this example?

This number obviously depends on the choice of algorithm/metric, the underlying data, and the use that will be made of the resulting clusters; a dataset could have 3 natural groups when seen through the lens of $k-$means, but only 2 clusters for a specific choice of parameter values in DBSCAN, and so on.

This problem could be overcome by producing clustering schemes (from the same family of algorithms) with an increasing number of clusters and to plot the average distance of a cluster member to its cluster representative (centroid) against the number of clusters. Any kink in the plot represents a number of clusters at which an increase does not provide an in-step increase in clustering "resolution", so to speak (see Figure 19.30 in the *Toy Example: Iris Dataset* subsection for an illustration).

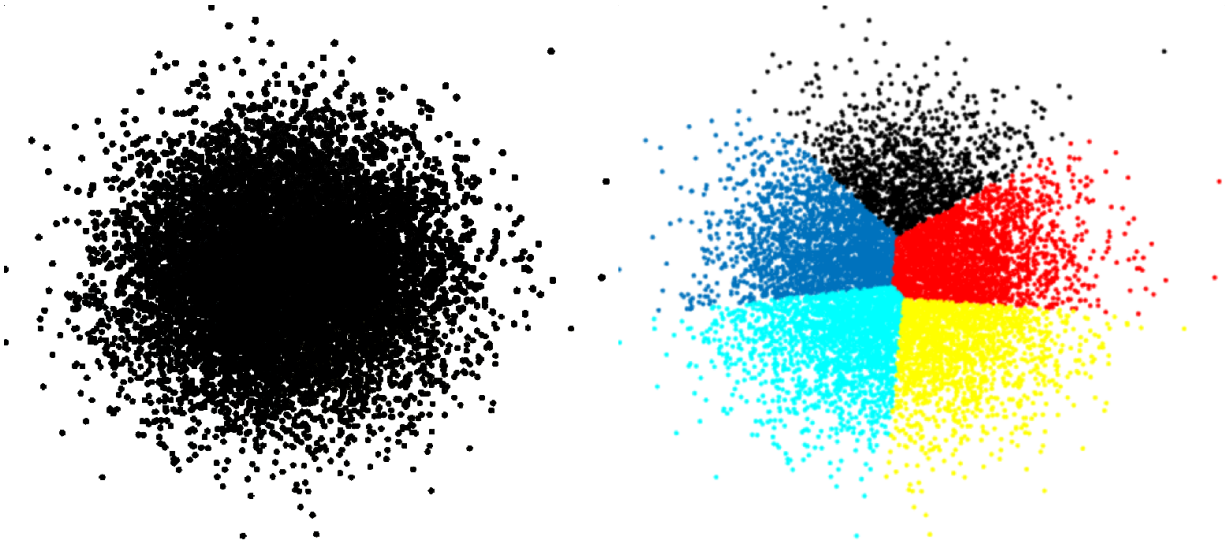


Figure 19.26: An illustration of ghost clustering with $k-$means, for $k = 5$.

And even when a cluster scheme has been accepted as valid, a **cluster description** might be difficult to come by – should clusters be described using representative instances or average values or some combination of its' members most salient features? Although there are exceptions, the ease with which clusters can be described often provides an indication about how **natural** the groups really are.

One of the most frustrating aspects of the process is that most methods will find clusters in the data even if there are none – although DBSCAN is exempt from this **ghost clustering** phenomenon (see Figure 19.26 for a $k-$means example).

Finally, analysts should beware (and resist) the temptation of ***a posteriori* rationalisation** – once clusters have been found, it is tempting to try to "explain" them; why are the groups as they have been found? But that is a job for domain experts, at best, and a waste of time and resources, at worst. Tread carefully.

19.5.4 Clustering Validation

What does it mean for a clustering scheme to be **better** than another? What does it mean for a clustering scheme to be **valid**? What does it mean for a single cluster to be **good**? **How many** clusters are there in the data, really?

These are not easy questions to answer. In general, asking if a clustering scheme is the right one or a good one is meaningless – much better to ask if it is **optimal** or **sub-optimal**, potentially in comparison to other schemes.

An **optimal** clustering scheme is one which

- **maximizes separation** between clusters;
- **maximizes similarity** within groups;
- agrees with the **human eye test**, and
- is **useful** at achieving its goals.

There are 3 families of **clustering validation** approaches:

- **external**, which use additional information (but the labels in question might have very little to do with the similarity of the observations);
- **internal**, which use only the clustering results (shape and size of clusters, etc), and
- **relative**, which compare across a number of clustering attempts.

In order to illustrate some of the possibilities, consider a dataset with **clustering scheme** $\mathscr{C} = \{\mathscr{C}_1, \ldots, \mathscr{C}_N\}$, where $\mathscr{C}_m$'s centroid is denoted by c_m, and the average distance of $\mathscr{C}_m$'s members to c_m is denoted by s_m. The **Davies-Bouldin Index** is defined as

$$\text{DB}_{\mathscr{C}} = \frac{1}{N} = \sum_{i=1}^{N} \max_{j \neq i} \left\{ \frac{s_i + s_j}{d(c_i, c_j)} \right\},$$

where d is the selected distance metric. Since $\text{DB}_{\mathscr{C}}$ is only defined using the clustering results, it is an **internal validation method**.

Heuristically, if the **cluster separation is small**, we might expect $d(c_i, c_j)$ to be (relatively) small, and so $\text{DB}_{\mathscr{C}}$ should be (relatively) large. In the same vein, if the clusters are **heterogeneous**, we might expect $s_i + s_j$ to be (relatively) large, and so $\text{DB}_{\mathscr{C}}$ should be (relatively) large.

In short, when the clustering scheme is **sub-optimal**, $\text{DB}_{\mathscr{C}}$ is "large". This suggests another way to determine the optimal number of clusters – pick the scheme with minimal $\text{DB}_{\mathscr{C}}$ (see Figure 19.30, which uses a modified version of the index, for an illustration).

Other **cluster quality metrics** exist, including **SSE**, **Dunn's Index**, the **Silhouette Metric**, etc. [3, 14].

19.5.5 Case Study: Pittsburgh Livehoods

When we think of similarity at the urban level, we typically think in terms of neighbourhoods. Is there some other way to identify similar parts of a city?

In *The Livehoods Project: Utilizing Social Media to Understand the Dynamics of a City* [12], Cranshaw *et al.* study the social dynamics of urban living spaces with the help of clustering algorithms.

Objective The researchers aims to draw the boundaries of **livehoods**, areas of similar character within a city, by using clustering models. Unlike **static** administrative neighborhoods, the livehoods are defined based on the habits of people who live there.

Methodology The case study introduces **spectral clustering** to discover the **distinct geographic areas** of the city based on its inhabitants' collective **movement patterns**. Semi-structured interviews are also used to **explore**, **label**, and **validate** the resulting clusters, as well as the urban dynamics that shape them.

Livehood clusters are built using the following methodology:

1. a **geographic distance** is computed based on pairs of check-in venues' coordinates;
2. **social similarity** between each pair of **venues** is computed using cosine measurements,
3. spectral clustering produces **candidate livehoods** clusters;
4. interviews are conducted with residents in order to **validate** the clusters discovered by the algorithm.

Data The data comes from two sources, combining 11 million (a recommendation site for venues based on users' experiences) check-ins from the dataset of Chen et al. [10] and a new dataset of 7 million Twitter check-ins downloaded between June and December of 2011.

For each check-in, the data consists of the **user ID**, the **time**, the **latitude and longitude**, the **name of the venue**, and its **category**.

In this case study, it is livehood clusters from the city of Pittsburgh, Pennsylvania, that are examined *via* 42,787 check-ins of 3840 users at 5349 venues.

Strengths and Limitations of the Approach

- The technique used in this study is **agnostic** towards the particular source of the data: it is not dependent on meta-knowledge about the data.
- The algorithm may be prone to "majority" bias, consequently misrepresenting/hiding minority behaviours.
- The dataset is built from a **limited** sample of check-ins shared on Twitter and are therefore biased towards the types of visits/locations that people typically want to share **publicly**.
- Tuning the clusters is non-trivial: experimenter bias may combine with "confirmation bias" of the interviewees in the validation stage – if the researchers are themselves residents of Pittsburgh, will they see clusters when there are none?

Procedures The Livehoods project uses a **spectral clustering model** to provide structure for local **urban areas** (UAs), grouping close Foursquare venues into clusters based on both the **spatial proximity** between venues and the **social proximity** which is derived from the distribution of people that check-in to them.

The guiding principle of the model is that the "character" of an UA is defined both by the types of venues it contains and by the people frequent them as part of their daily activities. These clusters are referred to as **Livehoods**, by analogy with more traditional neighbourhoods.

Let V be a list of Foursquare venues, A the associated **affinity matrix** representing a measure of similarity between each venue, and $G_m(A)$ be the graph obtained from the A by linking each venue to its nearest m neighbours. Spectral clustering is implemented as follows:[38]

1. Compute the diagonal degree matrix $D_{ii} = \sum_j A_{ij}$;
2. Set the Laplacian matrix $L = D - A$ and

$$L_{\text{norm}} = D^{-1/2} L D^{-1/2};$$

38: We will discuss spectral clustering and other clustering algorithms in detail in Chapter 22, *Spotlight on Clustering*.

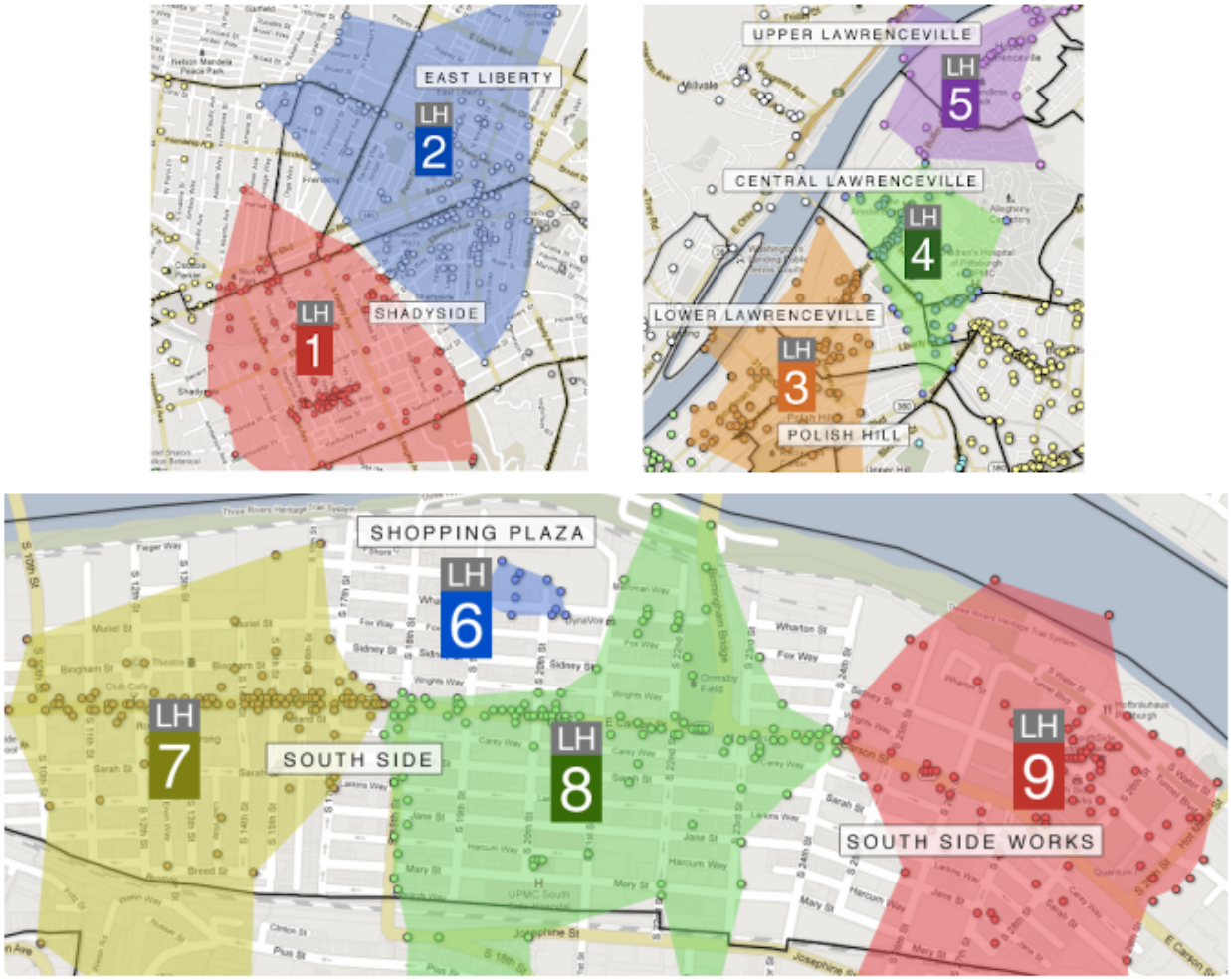


Figure 19.27: Some livehoods in metropolitan Pittsburgh, PA: Shadyside/East Liberty, Lawrenceville/Polish Hill, and South Side. Municipal borders are shown in black.

3. Find the k smallest eigenvalues of L_{norm}, where k is the index which provides the biggest jump in successive eigenvalues of eigenvalues of L_{norm}, in increasing order;
4. Find the eigenvectors $e_1, \ldots e_k$ of L corresponding to the k smallest eigenvalues;
5. Construct the matrix E with the eigenvectors $e_1, \ldots e_k$ as columns;
6. Denote the rows of E by $y_1, \ldots, y_n$, and cluster them into k clusters $C_1, \ldots, C_k$ using k-means. This induces a clustering $\{A_1, \ldots, A_k\}$ defined by
$$A_i = \{j \mid y_j \in C_i\};$$
7. For each A_i, let $G(A_i)$ be the subgraph of $G_m(A)$ induced by vertex A_i; split $G(A_i)$ into connected components; add each component as a new cluster to the list of clusters, and remove the subgraph $G(A_i)$ from the list;
8. Let b be the area of bounding box containing coordinates in the set of venues V, and b_i be the area of the box containing A_i; if $\frac{b_i}{b} > \tau$, delete cluster A_i, and redistribute each of its venues $v \in A_i$ to the closest A_j under the distance measurement.

Results, Evaluation and Validation The parameters used for the clustering were $m = 10$, $k_{\min} = 30$, $k_{\max} = 45$, and $\tau = 0.4$. The results for three areas of the city are shown in Figure 19.27. In total, 9 livehoods have been identified and validated by 27 Pittsburgh residents; the article has more information on this process.

- **Municipal Neighborhoods Borders:** livehoods are dynamic, and evolve as people's behaviours change, unlike the fixed neighbourhood borders set by the city government.
- **Demographics:** the interviews displayed strong evidence that the demographics of the residents and visitors of an area often play a strong role in explaining the divisions between livehoods.
- **Development and Resources:** economic development can affect the character of an area. Similarly, the resources (or lack there of) provided by a region has a strong influence on the people that visit it, and hence its resulting character. This is assumed to be reflected in the livehoods.
- **Geography and Architecture:** the movements of people through a certain area is presumably shaped by its geography and architecture; livehoods can reveal this influence and the effects it has over visiting patterns.

Take-Away While this is a neat example of practical clustering, its main take-away, from our perspective, is to remind everyone that $k-$means is not the sole clustering algorithm in applications!

19.5.6 Toy Example: Iris Dataset

Iris is a genus of plants with showy flowers. The `iris` dataset contains 150 observations of 5 attributes for specimens collected by Anderson, mostly from a Gaspé peninsula's pasture in the 1930s [19].[39]

The attributes are:

- **petal width**
- **petal length**
- **sepal width**
- **sepal length**
- **species** (virginica, versicolor, setosa)

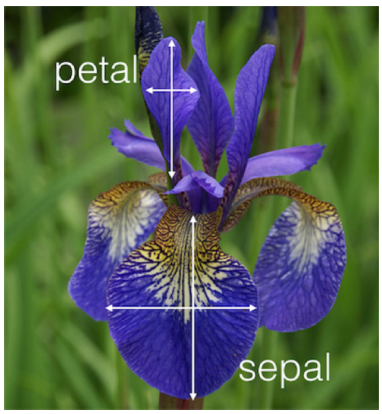
This dataset has become synonymous with data analysis, being used to showcase just about every algorithm under the sun. That is, sadly, also what we are going to do in this section.[40]

A **principal component projection** of the dataset, with species indicated by colours, is shown in Figure 19.28 (left).

From an **unsupervised learning** point of view, one question of interest is whether the observations form natural groupings, and, if so, whether these groupings correspond to the (known) species.

We use the $k-$means algorithm with Euclidean distance to resolve the problem. Since we do not know how many clusters there should be in the data (the fact that there are 3 species does not mean that there should be 3 clusters), we run 40 replicates for $k = 2, \ldots, 15$.

39:



40: Note that the iris dataset has started being phased out in favour of the penguin dataset [47], for reasons that do not solely have to do with its overuse (hint: take a look at the name of the journal that published Fisher's paper).

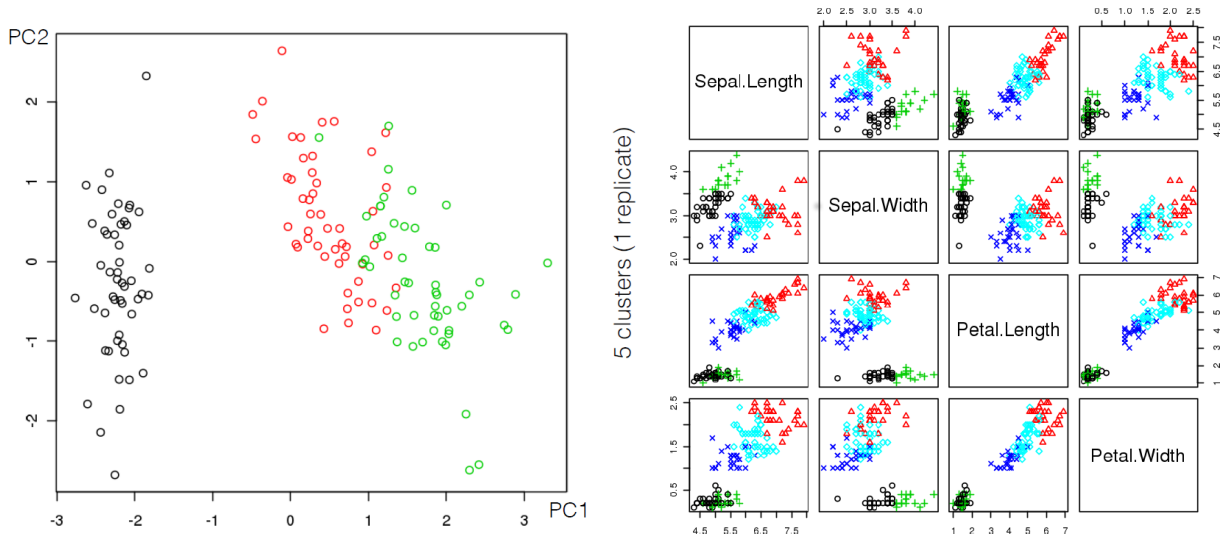


Figure 19.28: Classification of the iris dataset's 3 species, projected on the first 2 principal components (left); optimal clustering results for the iris dataset – one replicate, $k = 5$ (right).

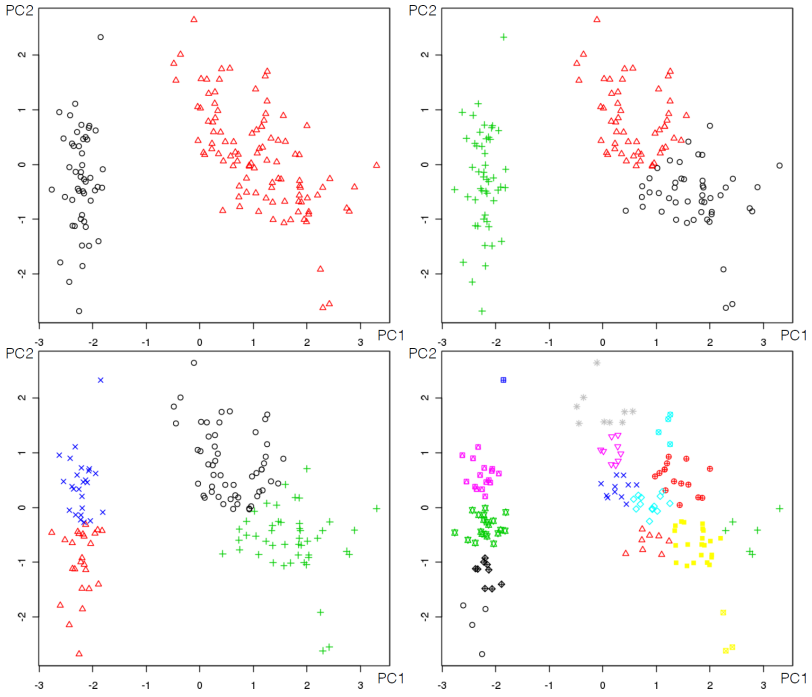


Figure 19.29: Clustering results on the iris dataset with $k-$means, for $k = 2, 3, 4, 15$ (from left to right).

For each replicate, we compute a (modified) **Davies-Bouldin Index** and the **Sum of Squared Errors** of the associated clustering schemes (see Figure 19.30 for the output) – the validation curves seem to indicate that there could be either 3 of 5 natural $k-$means clusters in the data. Is this a surprising outcome?

A single replicate with $k = 5$ is shown in Figure 19.28 (right). Would you consider this representative final clustering scheme to be meaningful? We show how to obtain these clustering results *via* `R` in Section 19.7 (*Clustering: Iris Dataset*).

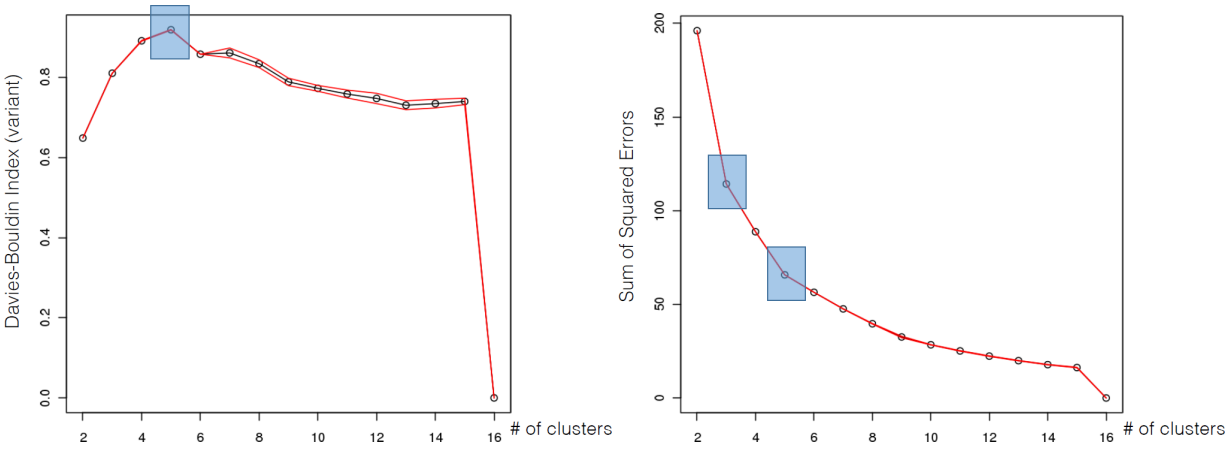


Figure 19.30: Optimal clustering results for the iris dataset: 5 clusters using (modified) Davies-Bouldin index and Sum of Squared Errors.

19.6 Issues and Challenges

> "We all say we like data, but we don't. We like getting insight out of data. That's not quite the same as liking data itself. In fact, I dare say that I don't quite care for data, and it sounds like I'm not alone." [38]

19.6.1 Bad Data

The main difficulties with data is that it is not always **representative** of the situation that we would like to model and that it might not be **consistent** (the collection and collation methods may have changed over time, say). There are other potential data issues [38]:

- the data might be formatted for human consumption, not machine readability;
- the data might contain lies and mistakes;
- the data might not reflect reality, and
- there might be additional sources of bias and errors (imputation bias, replacing extreme values with average values, proxy reporting, etc.).

Seeking perfection in the data beyond a "reasonable" threshold[41] can hamper the efforts of analysts: different quality requirements exist for academic data, professional data, economic data, government data, military data, service data, commercial data, etc. It can be helpful to remember the engineering dictum: "close enough is good enough"![42] The challenge lies in defining what is "close enough" for the application under consideration.

41: This threshold is difficult to establish exactly, however.

42: In terms of completeness, coherence, correctness, and accountability.

Even when all (most?) data issues have been mitigated, there remains a number of common **data analysis pitfalls**:

- analyzing data **without understanding the context**;
- using **one and only one tool** (by choice or by fiat) – neither the "cloud", nor Big Data, nor Deep Learning, nor Artificial Intelligence will solve all of an organization's problems;

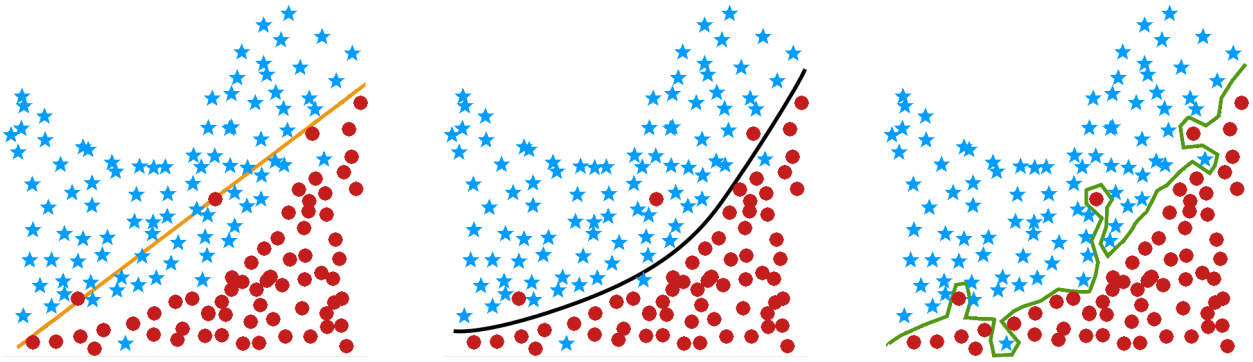


Figure 19.31: Illustration of underfitting (left) and overfitting (right) for a classification task – the optimal classifier (middle) might reach a compromise between accuracy and simplicity.

- analyzing data just for the sake of analysis,
- having **unrealistic expectations** of data analysis/DS/ML/AI – in order to optimize the production of actionable insights from data, we must first recognize the methods' domains of application and their limitations.

19.6.2 Overfitting/Underfitting

In a traditional statistical model, $p-$values and goodness-of-fit statistics are used to validate the model. But such statistics cannot always be computed for predictive data science models. We recognise a "good" model based on how well **it performs on unseen data**.

In practice, training sets and ML methods are used to search for **rules** and **models** that are **generalizable to new data** (or validation/testing sets).

Problems arise when knowledge that is gained from supervised learning does not generalize properly to the data. Ironically, this may occur if the rules or models **fit the training set too well** – in other words, the results are too specific to the training set (see Figure 19.31 for an illustration of **overfitting** and **underfitting**).

A simple example may elucidate further. Consider the following rules regarding hair colour among humans:

- **vague rule** – some people have black hair, some have brown hair, some blond, and some red;[43]
- **reasonable rule** – in populations of European descent, approximately 45% have black hair, 45% brown hair, 7% blond and 3% red, and
- **overly specific rule** – in every 10,000 individuals of European descent, we predict there are 46.32% with black hair, 47.27% with brown hair, 6.51% with blond hair, and 0.00% with red hair.[44]

With the overly specific rule, we would predict that there are no redheads in populations of European descent, which is false. This rule is **too specific** to the particular training subset that was used to produce it.[45]

More formally, underfitting and overfitting can be viewed as resulting from the **level of model complexity** (see Figure 19.32).

43: This is obviously "true", but too general to be useful for predictions.

44: This rule presumably emerges from redhead-free training data.

45: We could argue that the data was simply not representative – using a training set with redheads would yield a rule that would make better predictions. But "over-reporting/overconfidence" (which manifest themselves with the use of significant digits) is also part of the problem.

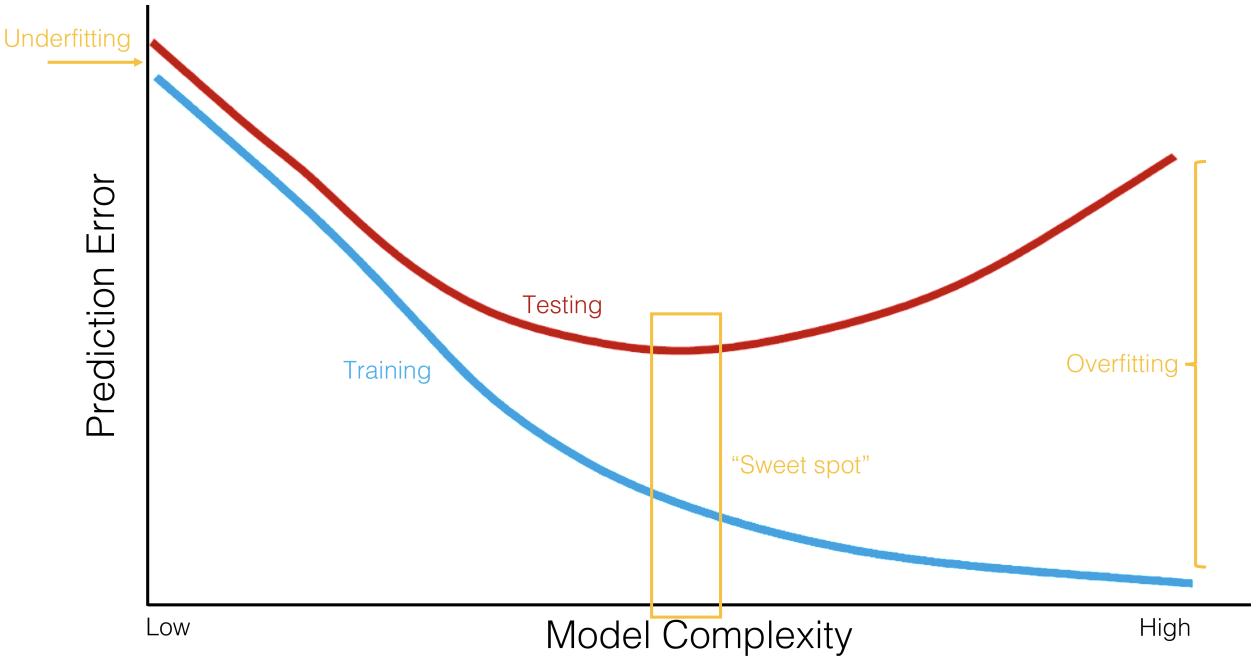


Figure 19.32: Underfitting and overfitting as a function of model complexity; error prediction on training sample (blue) and testing sample (red). High error prediction rates for simple models are a manifestation of underfitting; large difference between error prediction rates on training and testing samples for complex models are a manifestation of overfitting. Ideally, model complexity is chosen to reach the situation's 'sweet spot'; fishing for the ideal scenario might diminish explanatory power (based on [24]).

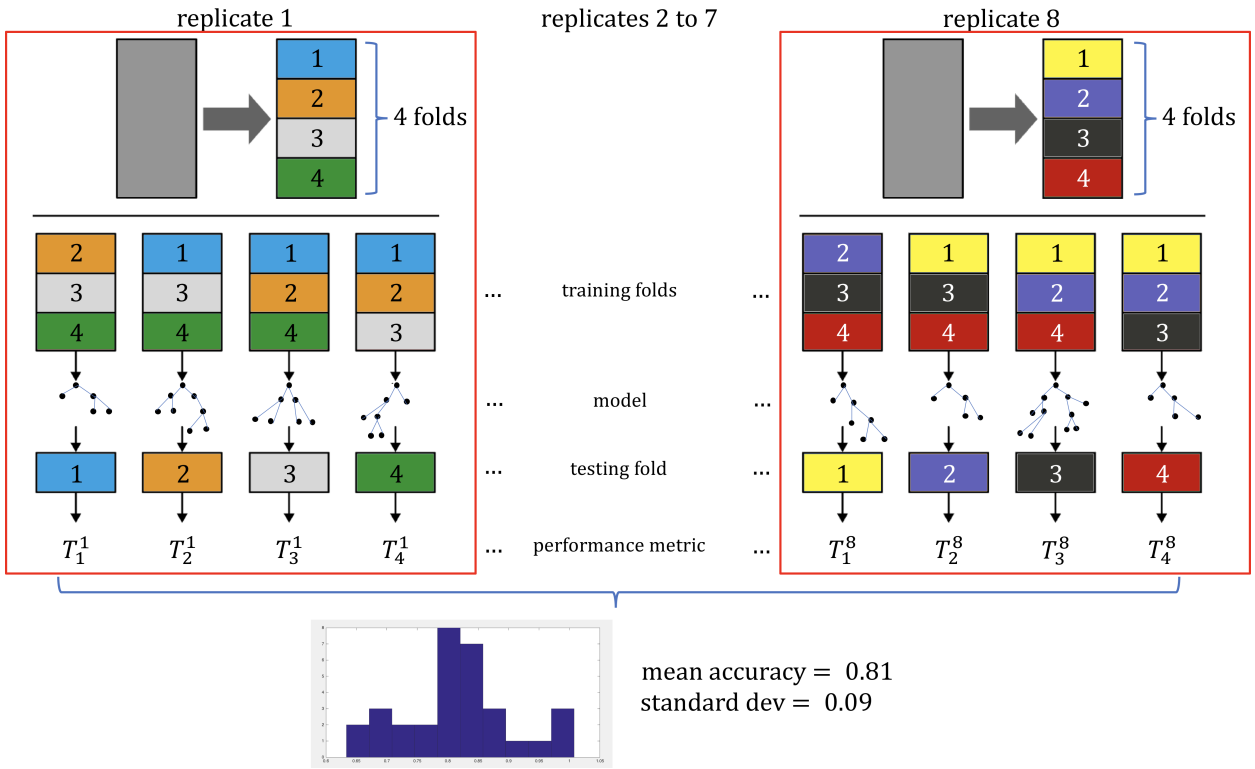


Figure 19.33: Schematic illustration of cross-fold validation, for 8 replicates and 4 folds; $8 \times 4 = 32$ models from a given family are built on various training sets (consisting of 3/4 of the available data – the training folds). Model family performance is evaluated on the respective holdout folds; the distribution of the performance metrics (in practice, some combination of the mean/median and standard deviation) can be used to compare various model families (based on [46, 55]).

Underfitting can be overcome by using more complex models (or models that use a larger proportion of a dataset's variables). Overfitting, on the other hand, can be overcome in several ways:

- **using multiple training sets** (ensemble learning approaches), with overlap being allowed – this has the effect of reducing the odds of finding spurious patterns based on quirks of the training data;
- **using larger training sets** may also remove signal which is too specific to small training sets: a 70%/30% split is often suggested, and
- **using simpler models** (or models that use a dataset with a reduced number of variables as input).

When using multiple training sets, the size of the dataset may also affect the suggested strategy: when faced with

- **small datasets** (less than a few hundred observations, say, but that depends on numerous factors such as computer power and number of tasks), use 100-200 repetitions of a **bootstrap procedure** [27];
- **average-sized datasets** (less than a few thousand observations), use a few repetitions of 10-fold cross-validation [27, 55] (see Figure 19.33 for an illustration), and
- **large datasets**, use a few repetitions of a holdout split (70%/30%?).

No matter which strategy is eventually selected, the machine learning approach requires ALL models to be evaluated on **unseen data**.[46]

46: These issues will be revisited in Chapters 20 (*Regression and Value Estimation*) and 21 (*Spotlight on Classification*).

19.6.3 Appropriateness and Transferability

Data science models will continue to be used heavily in the near future; while there are pros and cons to their use on ethical and other non-technical grounds, their applicability is also driven by **technical considerations**.

DS/ML/AI methods are **not** appropriate if:

- existing (legacy) datasets absolutely must be used instead of ideal/appropriate datasets;[47]
- the dataset has attributes that usefully predict a value of interest, but these attributes **are not available** when a prediction is required (e.g. the total time spent on a website may be predictive of a visitor's purchases, but the prediction must be made before the total time spent on the website is known), and
- class membership or numerical outcome is going to be predicted using an unsupervised learning algorithm.[48]

47: "It's the best data we have!" does not mean that it is the right data, or even good data.

48: For instance, clustering loan default data might lead to a cluster contains many defaulters – if new instances get added to this cluster, should they automatically be viewed as loan defaulters?

Every model makes certain assumptions about what is and is not **relevant** to its workings, but there is a tendency to only gather data which is **assumed** to be relevant to a particular situation. If the data is used in other contexts, or to make predictions depending on attributes for which no data is available, then there might be no way to **validate the results**.[49] This is not just an esoteric consideration: **over-generalizations and inaccurate predictions can lead to harmful results**.

49: For instance, can we use a model that predicts whether a borrower will default on a mortgage or not to also predict whether a borrower will default on a car loan or not? The problem is compounded by the fact that there might be some link between mortgage defaults and car loan defaults, but the original model does not necessarily takes this into account.

19.6.4 Myths and Mistakes

We end this section by briefly repeating various **data science myths**, originally found in [37]:

1. DS is about algorithms;
2. DS is about predictive accuracy;
3. DS requires a data warehouse;
4. DS requires a large quantity of data, and
5. DS requires only technical experts,

as well as common **data analysis mistakes** [same source]:

1. selecting the wrong problem;
2. getting by without metadata understanding;
3. not planning the data analysis process;
4. insufficient business/domain knowledge;
5. using incompatible data analysis tools;
6. using tools that are too specific;
7. favouring aggregates over individual results;
8. running out of time;
9. measuring results differently than the client, and
10. naïvely believing what one is told about the data.

It remains the analyst's and/or the consultant's responsibility to address these issues with the stakeholders and/or clients, **the earlier, the better**. It is safer to assume that not everyone is on the same page – prod and ask, early and often.

19.7 R Examples

We provide the R code that was used to produce the outputs of the toy examples in Sections 19.3, 19.4, and 19.5.

19.7.1 ARM: Titanic

This example refers to the Titanic dataset toy example of Section 19.3. The very first step in programming withRis to import data.

Setting up the Titanic dataset

```
class = as.factor(c(rep("3rd",52),rep("1st",118),
   rep("2nd",154),rep("3rd",387),rep("Crew",670),
   rep("1st",4),rep("2nd",13.01),rep("3rd",89),
   rep("Crew",3),rep("1st",5),rep("2nd",11),
   rep("3rd",13),rep("1st",1),rep("2nd",13),
   rep("3rd",14),rep("1st",57),rep("2nd",14),
   rep("3rd",75),rep("Crew",192),rep("1st",140),
   rep("2nd",80),rep("3rd",76),rep("Crew",20)))
sex = as.factor(c(rep("Male",35),rep("Female",17),
   rep("Male",1329),rep("Female",109),rep("Male",29),
   rep("Female",28),rep("Male",338),rep("Female",316)))
```

```
age = as.factor(c(rep("Child",52),rep("Adult",1438),
   rep("Child",57),rep("Adult",654)))
survived = as.factor(c(rep("No",1490),rep("Yes",711)))
titanic = data.frame(class,sex,age,survived)
```

We briefly explore the structure of data.

Summary data

```
summary(titanic)
table(titanic$age,titanic$survived)
table(titanic$class,titanic$survived)
```

class	sex	age	survived
1st: 325	Female: 470	Adult: 2092	No: 1490
2nd: 285	Male: 1731	Child: 109	Yes: 711
3rd: 706			
Crew: 885			

age/survived	No	Yes	class/survived	No	Yes
Adult	1438	654	**1st**	122	203
Child	52	57	**2nd**	167	118
			3rd	528	178
			Crew	673	212

Then we use the `arules` package function `apriori()`, which returns all possible rules (built by the apriori algorithm). By default, `apriori()` creates rules with minimum support of 0.1, minimum confidence of 0.8, and maximum of 10.

Original apriori rules

```
rules.titanic <- arules::apriori(titanic)
```

```
Apriori parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen
        0.8    0.1    1 none FALSE            TRUE       5     0.1      1
 maxlen target  ext
     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 220

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[10 item(s), 2201 transaction(s)] done [0.00s].
sorting and recoding items ... [9 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
```

```
checking subsets of size 1 2 3 4 done [0.00s].
writing ... [27 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

For a rule $X \rightarrow Y$, `arule` evaluates a number of default metrics. We can extract the rules using the function `inspect()`: there are 27 in total.

Inspecting the original rules

```
arules::inspect(rules.titanic)
```

```
     lhs                                    rhs           support   confidence
[1]  {}                                  => {age=Adult}   0.9504771 0.9504771
[2]  {class=2nd}                         => {age=Adult}   0.1185825 0.9157895
[3]  {class=1st}                         => {age=Adult}   0.1449341 0.9815385
...
[26] {class=Crew, sex=Male, survived=No}  => {age=Adult}   0.3044071 1.0000000
[27] {class=Crew, age=Adult, survived=No} => {sex=Male}    0.3044071 0.9955423

     coverage  lift      count
[1]  1.0000000 1.0000000 2092
[2]  0.1294866 0.9635051  261
[3]  0.1476602 1.0326798  319
...
[26] 0.3044071 1.0521033  670
[27] 0.3057701 1.2658514  670
```

We set our own parameters to create a new list of rules, with minimum support of 0.005 and minimum confidence of 0.8. As we are naturally interested in finding combinations of attributes associated with survival (either Yes or No), we only retain rule for which the conclusion is `{survived=No}` or `{survived=Yes}`.

There are 12 such rules.

Constrained apriori rules

```
rules.titanic.2 <- arules::apriori(titanic,
     parameter = list(minlen=2, supp=0.005, conf=0.8),
     appearance = list(rhs=c("survived=No","survived=Yes"),
                       default="lhs"),
     control = list(verbose=F))
```

We then sort the list by the lift value (in descending order) and print the results:

Sorting the constrained rules

```
rules.titanic.2 <- arules::sort(rules.titanic.2,
                      by=c("lift"), decreasing=TRUE)
```

Inspecting the constrained rules

```
arules::inspect(rules.titanic.2)
```

```
     lhs                                    rhs
[1]  {class=2nd, age=Child}              => {survived=Yes}
...
[11] {class=3rd, sex=Male, age=Adult}    => {survived=No}
[12] {class=3rd, sex=Male}               => {survived=No}

     support      confidence coverage    lift     count
[1]  0.010904134  1.0000000  0.010904134 3.095640  24
...
[11] 0.175829169  0.8376623  0.209904589 1.237379 387
[12] 0.191731031  0.8274510  0.231712858 1.222295 422
```

Are all these rules independent? If we know that

```
{class=3rd,sex=Male} => {survived=No}
```

is a rule, say, then we would not be surprised to find out that

```
{class=3rd,sex=Male,age=Adult} => {survived=No}
```

is also a rule.

The following chunk of code identifies which rules have an antecedent which is a subset of another rule's antecedent, marking one of them as **redundant**, and removing those from the set of rules, which brings us down to 8 rules:

Pruned apriori rules

```
subset.matrix <- as.matrix(arules::is.subset(
                 rules.titanic.2, rules.titanic.2))
subset.matrix[lower.tri(subset.matrix, diag=T)] <- NA
redundant.titanic <- colSums(subset.matrix, na.rm=T) >= 1
rules.titanic.2.pruned <- rules.titanic.2[!redundant.titanic]
arules::inspect(rules.titanic.2.pruned)
```

```
    lhs                                rhs            supp conf cove lift count
[1] {class=2nd, age=Child}          => {survived=Yes} 0.01 1.00 0.01 3.10  24
[2] {class=1st, sex=Female}         => {survived=Yes} 0.06 0.97 0.07 3.01 141
[3] {class=2nd, sex=Female}         => {survived=Yes} 0.04 0.88 0.05 2.72  93
[4] {class=Crew, sex=Female}        => {survived=Yes} 0.01 0.87 0.01 2.69  20
[5] {class=2nd, sex=Male, age=Adult} => {survived=No}  0.07 0.92 0.08 1.35 154
[6] {class=2nd, sex=Male}           => {survived=No}  0.07 0.86 0.08 1.27 154
[7] {class=3rd, sex=Male, age=Adult} => {survived=No}  0.18 0.84 0.21 1.24 387
[8] {class=3rd, sex=Male}           => {survived=No}  0.19 0.83 0.23 1.22 422
```

We have presented the "interesting" associations in tabular format, but there are a variety of graphical representations as well (available in package `arulesViz`), such as:

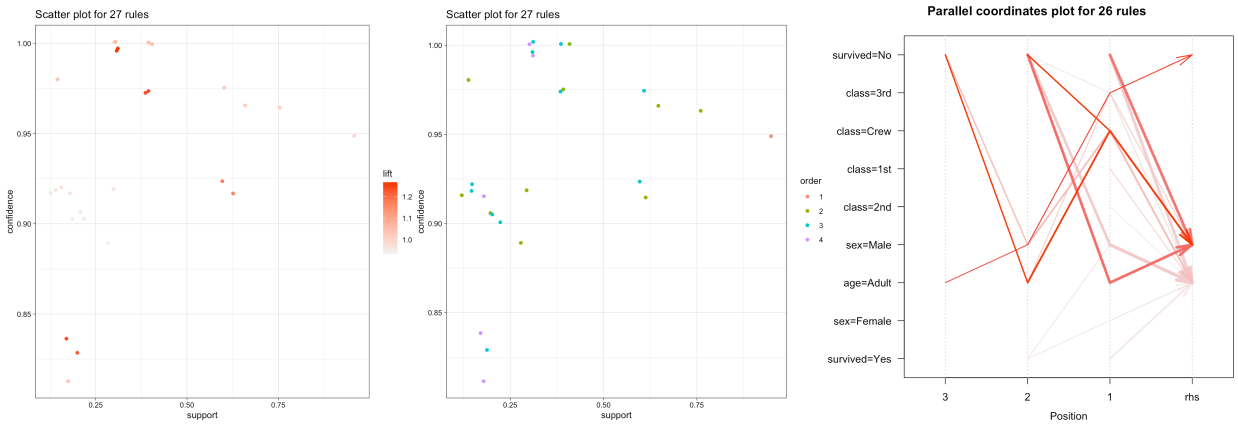
- a bubble chart;

- a two-key plot (taking into account the rules' lengths);
- a graph structure, or
- parallel coordinates (where the width of the arrows represents support and the intensity of the colour represent confidence).

The original rules are shown below:

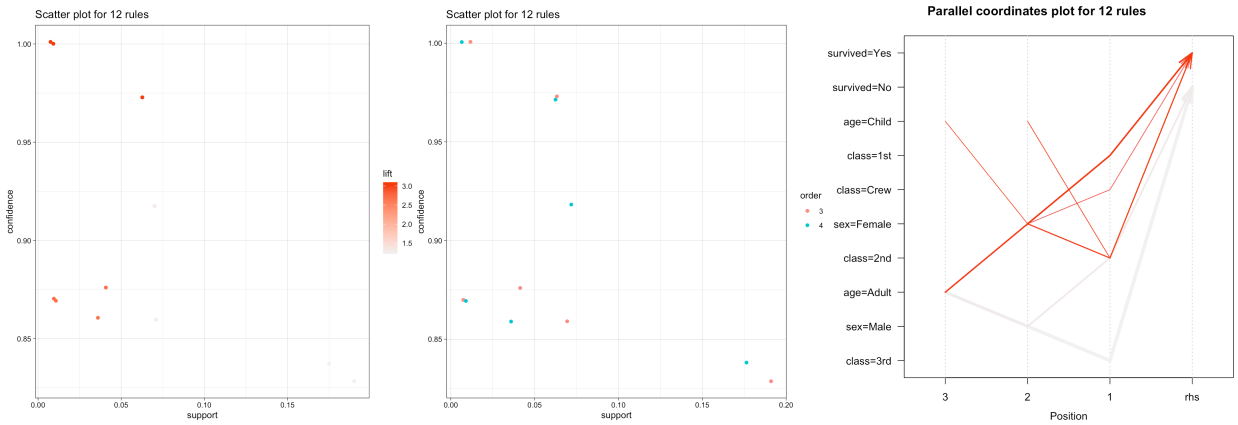
Visualizing the original rules

```
library(arulesViz)
plot(rules.titanic)
plot(rules.titanic, method="graph")
plot(rules.titanic, method="paracoord", control = list(reorder = TRUE))
```



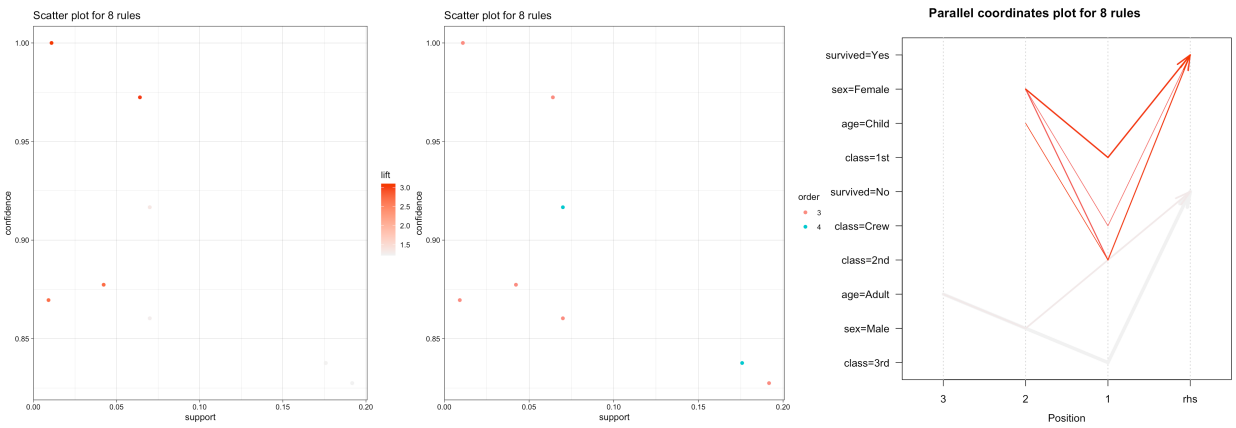
For the constrained and the pruned rules, we obtain:

```
plot(rules.titanic.2)
plot(rules.titanic.2, method="graph")
plot(rules.titanic.2, method="paracoord")
```




```
plot(rules.titanic.2.pruned)
plot(rules.titanic.2.pruned, method="graph")
plot(rules.titanic.2.pruned, method="paracoord")
```

Is anything surprising about these outcomes?



19.7.2 Classification: Kyphosis Dataset

This example refers to the Kyphosis dataset toy example of Section 19.4; we explore the built-in `kyphosis` dataset with two decision tree methods (`rpart()`, `ctree()`).

Let's get some information on the `kyphosis` dataset.

Getting the help file

```
?rpart::kyphosis
```

We can also determine its structure and summary statistics:

Kyphosis dataset structure

```
str(rpart::kyphosis)
```

```
'data.frame':   81 obs. of  4 variables:
 $ Kyphosis: Factor w/ 2 levels "absent","present": 1 1 2 ...
 $ Age     : int  71 158 128 2 1 1 61 37 113 59 ...
 $ Number  : int  3 3 4 5 4 2 2 3 2 6 ...
 $ Start   : int  5 14 5 1 15 16 17 16 16 12 ...
```

Summary data

```
summary(rpart::kyphosis)
```

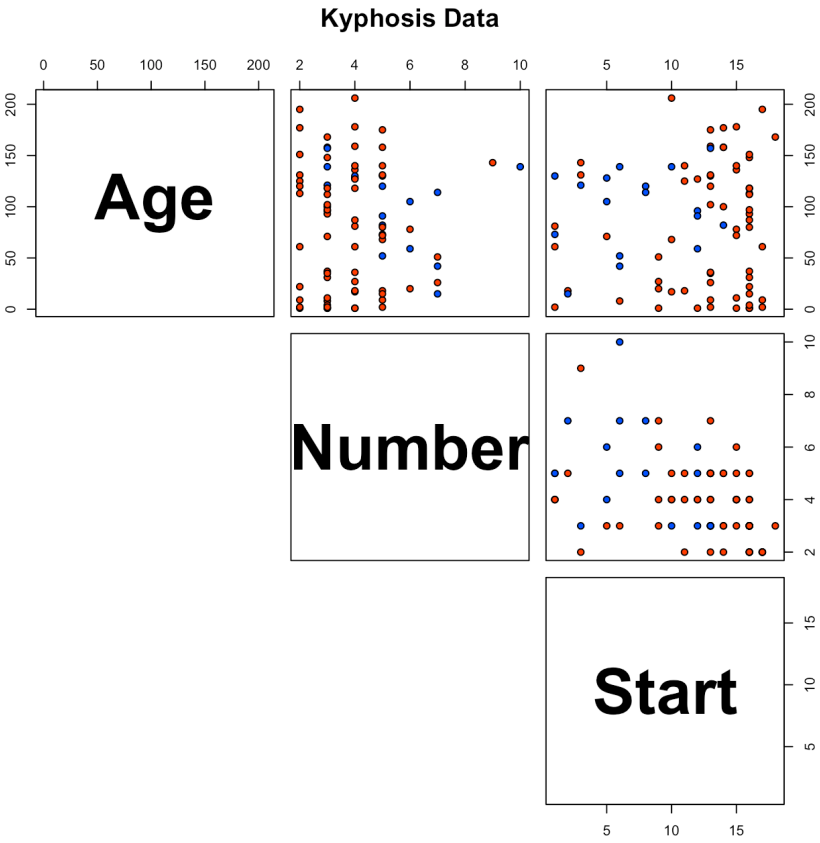
Kyphosis	Age	Number	Start
absent: 64	Min. : 1.00	Min. : 2.000	Min. : 1.00
present: 17	1st Qu.: 26.00	1st Qu.: 3.000	1st Qu.: 9.00
	Median : 87.00	Median : 4.000	Median : 13.00
	Mean : 83.65	Mean : 4.049	Mean : 11.49
	3rd Qu.: 130.00	3rd Qu.: 5.000	3rd Qu.: 16.00
	Max. : 206.00	Max. : 10.000	Max. : 18.00

As always, we should take the time to visualize the dataset. In this case, since there are 4 variables (one of which is categorical), a scatterplot matrix is probably a good approach.

Visualizing the kyphosis data

```
pairs(rpart::kyphosis[,2:4],
      main = "Kyphosis Data",
      bg = c("red", "blue")[unclass(rpart::kyphosis[,1])],
      pch = 21, lower.panel=NULL,
      cex.labels=4.5, labels=c("Age","Number","Start"),
      font.labels=2)
```

What should the legend of this scatterplot matrix be (**red**=?, **blue**=?).



We build a tree using the recursive partitioning algorithm implemented in `rpart`.[50] For the time being, we're assuming that the training set is the dataset as a whole (so there is no reason to expect that the decision trees should have predictive power, only descriptive power).

50: The package is called `rpart`, the function... also `rpart()`.

Building a recursive partition tree

```
set.seed(2) # for replicability
tree <- rpart::rpart(Kyphosis ~ Age + Number + Start,
                     method="class", data=rpart::kyphosis)
tree
```

```
n= 81

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 81 17 absent (0.79012346 0.20987654)
   2) Start>=8.5 62  6 absent (0.90322581 0.09677419)
     4) Start>=14.5 29  0 absent (1.00000000 0.00000000) *
     5) Start< 14.5 33  6 absent (0.81818182 0.18181818)
      10) Age< 55 12  0 absent (1.00000000 0.00000000) *
      11) Age>=55 21  6 absent (0.71428571 0.28571429)
        22) Age>=111 14  2 absent (0.85714286 0.14285714) *
        23) Age< 111 7  3 present (0.42857143 0.57142857) *
   3) Start< 8.5 19  8 present (0.42105263 0.57894737) *
```

We can access the method results by using `printcp()` (although how informative this output will prove depends on the expectations. . .)

Getting information about the tree

```
rpart::printcp(tree)
```

```
Classification tree:
rpart::rpart(formula = Kyphosis ~ Age + Number + Start, data = rpart::kyphosis,
    method = "class")

Variables actually used in tree construction:
[1] Age   Start

Root node error: 17/81 = 0.20988

n= 81

        CP nsplit rel error xerror    xstd
1 0.176471      0   1.00000 1.0000 0.21559
2 0.019608      1   0.82353 1.2353 0.23200
3 0.010000      4   0.76471 1.2941 0.23548
```

Details on the nodes and the splits can be obtained using `summary()`.

Summarizing the tree

```
summary(tree)
```

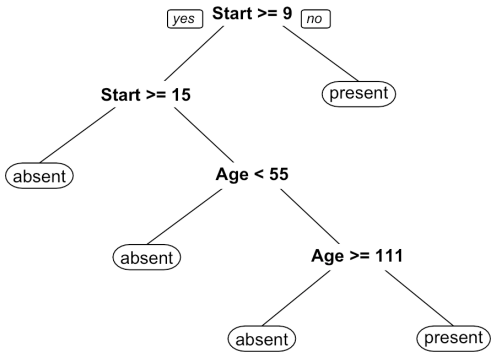
```
n= 81

          CP nsplit rel error   xerror      xstd
1 0.17647059      0 1.0000000 1.000000 0.2155872
2 0.01960784      1 0.8235294 1.235294 0.2320031
3 0.01000000      4 0.7647059 1.294118 0.2354756

Variable importance
 Start    Age Number
```

```
    64     24     12

Node number 1: 81 observations,    complexity param=0.1764706
  predicted class=absent   expected loss=0.2098765  P(node) =1
    class counts:    64    17
   probabilities: 0.790 0.210
  left son=2 (62 obs) right son=3 (19 obs)
  Primary splits:
      Start  < 8.5  to the right, improve=6.762330, (0 missing)
      Number < 5.5  to the left,  improve=2.866795, (0 missing)
      Age    < 39.5 to the left,  improve=2.250212, (0 missing)
  Surrogate splits:
      Number < 6.5  to the left,  agree=0.802, adj=0.158, (0 split)

...

Node number 23: 7 observations
  predicted class=present  expected loss=0.4285714  P(node) =0.08641975
    class counts:     3     4
   probabilities: 0.429 0.571
```

What are these nodes that are being referred to? Plotting the tree provides more information. Here is a basic plot:

Plotting the tree

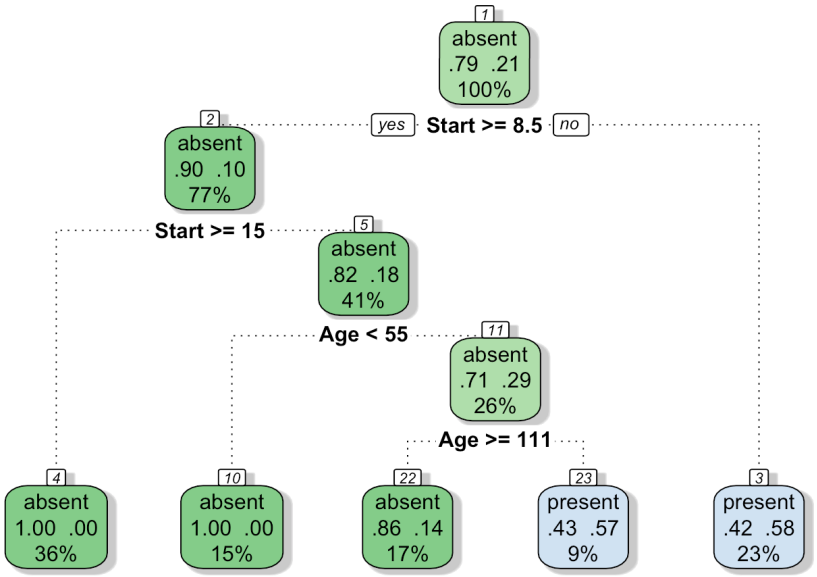
```
rpart.plot::prp(tree)
```



and a fancier one:

Plotting a fancier tree

```
rattle::fancyRpartPlot(tree,
          main="Classification Tree for Kyphosis")
```



In the fancy plot, does the **intensity** of the colour play a role? What about the **percentages**? What about the **decimals**?

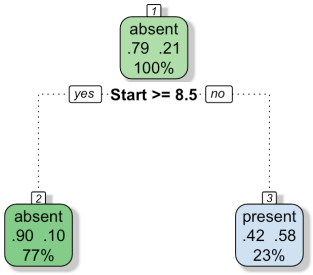
Unchecked tree growth usually leads to overfitting. This is typically a problem when datasets contain too many variables.[51] But overfitting is unlikely to be an issue in the case of the `kyphosis` dataset because it contains only three variables.

51: The corresponding decision tree will contain too many splits in that case.

Nevertheless, the code below shows you how you would **prune** the growth of the tree in general, by finding a value of `cp` which maximizes `xerror` (this will be revisited in Section 21.4.1, *Tree-Based Methods*).

Pruning and plotting the pruned tree

```
tree2 = rpart::prune(tree, cp = 0.02)
rattle::fancyRpartPlot(tree2)
```



How good is the classification model provided by the tree? We don't have access to $p-$values or confidence intervals – we need to rely on the model's **confusion matrix**.

We can obtain the predictions made by the model on the object tree by using the `predict()` function. This procedure takes each observation

and feeds it to the model, outputting the likelihood of kyphosis being `absent` or `present`.

Note that the probabilities are **calibrated** - compare with the Naive Bayes method which we will see later.

The following model predicts the class probabilities for all observations:

Getting the class probabilities

```
predictions1 = predict(tree, type = "prob")
```

The first 10 predictions are:

```
head(predictions1, 10)
```

absent	present
0.4210526	0.5789474
0.8571429	0.1428571
0.4210526	0.5789474
0.4210526	0.5789474
1.0000000	0.0000000
1.0000000	0.0000000
1.0000000	0.0000000
1.0000000	0.0000000
1.0000000	0.0000000
0.4285714	0.5714286

In general, the confusion matrix requires a specific prediction (`absent` or `present`), against which we compare the actual classification. Here, we have probabilities. How can we take the probabilities and transform them into specific predictions?

Here is one way to do this.

Building the confidence matrix

```
# uniformly generate a random number (between 0 and 1)
# for each of the observations
random1 <- runif(81)

# extract the actual classification of the observations
real <- rpart::kyphosis$Kyphosis

# join together the prediction probabilities,
# the random numbers, and the actual classification
# since we're joining together text and numbers,
# cbind coerces the factors to numerical values
# absent = 1, present = 2
test1 <- cbind(predictions1,random1,real)

# this code takes advantage of the numerical presentation
```

```
# of the factors to output a specific prediction
# if random1 < prob of absent, then
# real = absent (1), otherwise real = present (2)
pred1 <- 2-(test1[,3]<test1[,1])

# add the specific predictions to the test1 dataset
test1 <- cbind(test1,pred1)
```

The confusion matrix actually depends on the random variables generated in the previous chunk of code.

In this case, the first 10 predictions would be:

```
head(test1[,c(1,2,4,5)],10)
```

absent	present	real	pred1
0.4210526	0.5789474	1	1
0.8571429	0.1428571	1	1
0.4210526	0.5789474	2	2
0.4210526	0.5789474	1	2
1.0000000	0.0000000	1	1
1.0000000	0.0000000	1	1
1.0000000	0.0000000	1	1
1.0000000	0.0000000	1	1
1.0000000	0.0000000	1	1
0.4285714	0.5714286	2	1

We can now build the confusion matrix on the model predictions using `real` (actual classification) and `pred1` (specific model prediction).

The function `table()` produces a joint distribution, where the rows correspond to the first variable in the call (actual), and the columns to the second variable (predicted).

Confusion matrix

```
table(test1[,4],test1[,5])
```

```
   1  2
1 59  5
2  9  8
```

Note that the confusion matrix could be different every time a new set of predictions are made. Why would that be the case?

Is this a good classification model or not?

In this example, we computed the confusion matrix using the **entire dataset**. In a sense, we should not have been surprised that the results were decent, because we were using the same data to build the model and to evaluate it.

From a predictive perspective, classification models are built on a subset of the data (the **training set**) and evaluated on the remaining data (the

testing set). The idea is that if there IS a strong classification signal, it should be found in any representative subset. As a rule, we look for training sets making up between 70% and 80% of the data. They should be selected randomly.

We start by creating a training set with 50 instances, and we fit the `rpart()` algorithm to this data:[52]

52: If we use all variables, we do not need to specify them in the model call; we only need to use the "."

Training a recursive partition tree

```
sub <- c(sample(1:81, 50))
(fit <- rpart::rpart(Kyphosis ~ ., data = rpart::kyphosis,
                     subset = sub))
```

```
n= 50

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 50 9 absent (0.8200000 0.1800000)
  2) Start>=8.5 40 3 absent (0.9250000 0.0750000) *
  3) Start< 8.5 10 4 present (0.4000000 0.6000000) *
```

The confusion matrix has to be built on the testing set.[53] There are 2 ways to make a prediction: we either use the most likely outcome, or we generate a random vector of predictions using the probabilities for each class, as above.

53: Iindicated here by `-sub`, that is, the opposite of the `sub` indices.

We can predict on the basis of class:

Predictions on the testing set

```
table(predict(fit, rpart::kyphosis[-sub,], type = "class"),
              rpart::kyphosis[-sub, "Kyphosis"])
```

```
         absent present
  absent      19       3
  present      4       5
```

Or we can predict on the basis of probability:

```
prob.fit <- predict(fit, rpart::kyphosis[-sub,],
                    type = "prob")
random1 <- runif(31)
real <- rpart::kyphosis[-sub,"Kyphosis"]

# absent = 1, present = 2
test1 <- cbind(prob.fit,random1,real)
pred1 <- 2-(test1[,3]<test1[,1])
test1 <- cbind(test1,pred1)
table(test1[,4], test1[,5])
```

```
   1  2
1 20  3
2  5  3
```

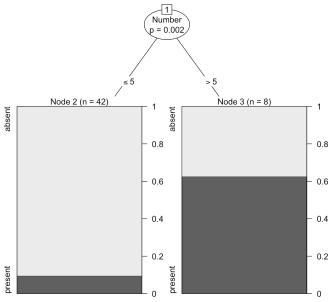
These are the confusion matrices that should be used to evaluate the decision's tree performance.

Another way to build decision trees is *via* `party`'s `ctree()` function. Conditional inference trees have the property that they will **automatically prune** themselves once a statistical criterion is met by the tree as a whole. The downside is that they do not usually pick fully reasonable splits (they may not conform to contextual understanding).

Building and plotting a conditional inference tree

```
kyphosis.ctree <- party::ctree(Kyphosis ~ .,
        data = rpart::kyphosis, subset = sub)
kyphosis.ctree
plot(kyphosis.ctree)
```

```
     Conditional inference tree with 2 terminal nodes

Response:  Kyphosis
Inputs:  Age, Number, Start
Number of observations:  50

1) Number <= 5; criterion = 0.998, statistic = 11.641
  2)*  weights = 42
1) Number > 5
  3)*  weights = 8
```



A very simple tree, as can be seen.

Confusion matrix on the testing set

```
table(predict(kyphosis.ctree, rpart::kyphosis[-sub,]),
              rpart::kyphosis[-sub,1])
```

```
         absent present
absent       21       7
present       2       1
```

How good of a model is this one? Which of the `rpart()` or `ctree()` model is preferable? Does this depend on the training set?

19.7.3 Clustering: Iris Dataset

This example refers to the Iris dataset toy example of Section 19.5; we cluster the ubiquitous (built-in) `iris` dataset, *via* k-means.

The procedure is straightforward:

1. cluster with $n = 2, \ldots, 15$ clusters;
2. display the **Within Sum of Squares** curve, as a function of the # of clusters;
3. display the **Davies-Bouldin curve**, as a function of the # of clusters,
4. select the optimal number of clusters on the basis of these curves.

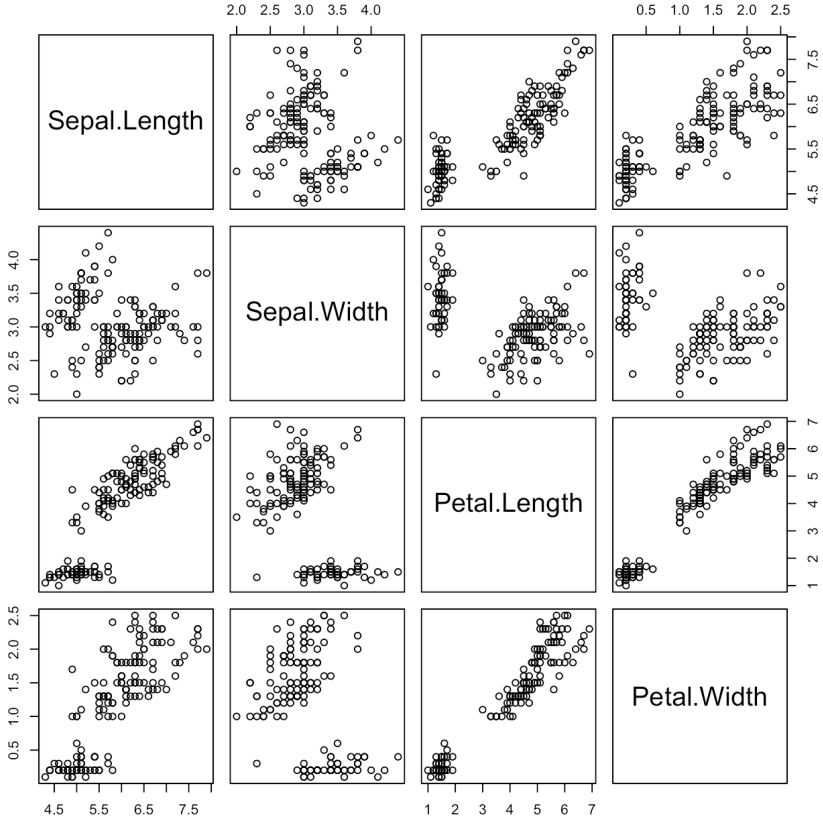
Let us load the data and take a look at the iris dataset.[54]

54: Without the species labels, as this is an unsupervised problem.

Visualizing the iris data

```
my.data<-iris[,1:4]
head(my.data)
pairs(my.data)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
1	5.1	3.5	1.4	0.2
2	4.9	3.0	1.4	0.2
3	4.7	3.2	1.3	0.2
4	4.6	3.1	1.5	0.2
5	5.0	3.6	1.4	0.2
6	5.4	3.9	1.7	0.4



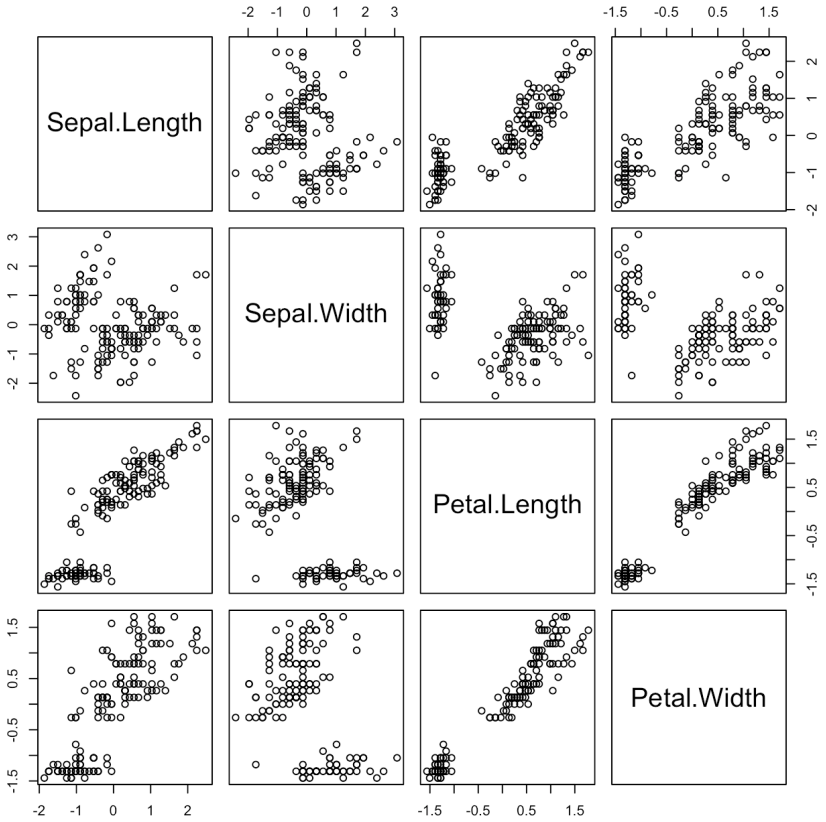
The eye test would most likely identify 2 clusters.

In preparation for the cluster analysis, we will scale the data so that all the variables are represented on the same scale. This can be done using the `scale()` function.

Scaling the data

```
my.data.scaled<-scale(my.data)
head(my.data.scaled)
pairs(my.data.scaled)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
1	-0.8976739	1.01560199	-1.335752	-1.311052
2	-1.1392005	-0.13153881	-1.335752	-1.311052
3	-1.3807271	0.32731751	-1.392399	-1.311052
4	-1.5014904	0.09788935	-1.279104	-1.311052
5	-1.0184372	1.24503015	-1.335752	-1.311052
6	-0.5353840	1.93331463	-1.165809	-1.048667



The "shape" of the dataset is the same, but the axis ranges have changed.

One way to reduce the dimension of the problem is to work with the **principal components** (see Chapter 23, *Feature Selection and Dimension Reduction*). The hope is that most of the variation in the data can be explained by a smaller number of derived variables, expressed as linear combinations of the original variables.

This can be accomplished in R with the `princomp()` function.

Principal Components

```
pc.agg.data = princomp(my.data.scaled)
summary(pc.agg.data)
```

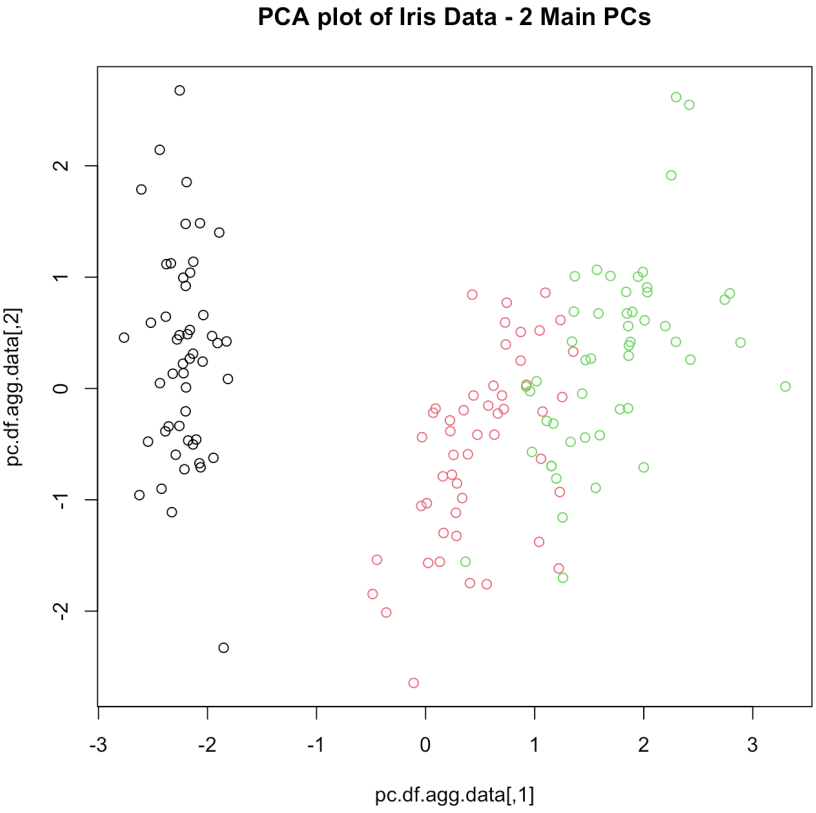
```
Importance of components:
                          Comp.1    Comp.2     Comp.3      Comp.4
Standard deviation     1.7026571 0.9528572 0.38180950 0.143445939
Proportion of Variance 0.7296245 0.2285076 0.03668922 0.005178709
Cumulative Proportion  0.7296245 0.9581321 0.99482129 1.000000000
```

This provides a summary of the "strength" of the signal in each component. The cumulative proportion of the variance is the value of interest. If 2 principal components are needed to explain 95% of the variance, we would expect that roughly 95% of the set is "2-dimensional".

As this is indeed the case, we opt to cluster the data projected on only the first 2 components, which can be accessed via the scores attribute of `pc.agg.data`. The plot below shows a 2D representation of the iris dataset obtained *via* principal components analysis (PCA).

Visualizing the PCA data

```
pc.df.agg.data = cbind(pc.agg.data$scores[,1],
                       pc.agg.data$scores[,2])
plot(pc.df.agg.data,col=iris[,5])
title('PCA plot of Iris Data - 2 Main PCs')
```



The **Davies-Bouldin** (DB) index is a measure that is used to determine the optimal number of clusters in the data. For a given dataset, the optimal number of clusters is obtained by maximizing the DB index (there are different versions of the DB index, all giving equivalent results).

```
Davies.Bouldin <- function(A, SS, m) {
  # A  - the clusters' centres
  # SS - the within sum of squares
  # m  - the sizes of the clusters
  N <- nrow(A)   # number of clusters

  # intercluster distance
  S <- sqrt(SS/m)

  # Get the distances between centres
  M <- as.matrix(dist(A))

  # Get the ratio of intercluster/centre.dist
  R <- matrix(0, N, N)
  for (i in 1:(N-1)) {
    for (j in (i+1):N) {
      R[i,j] <- (S[i] + S[j])/M[i,j]
      R[j,i] <- R[i,j]
    }
  }
  return(mean(apply(R, 1, max)))
}
```

But we do not yet know how many clusters we will ultimately be using,[55] so we will construct $k-$means clusters for a variety of values k.

We will in fact produce 40 replicates for each $k = 2, \ldots, 15$ and track both the DB index and the **Within Sums of Squares** (SS), which is a measure of how similar observations are within each cluster, and how different they are from observations in other clusters.[56]

If the DB index does not provide a clear-cut winner, the optimal number of clusters is obtained when the slope of the SS curve flattens "drastically".[57]

We start by setting up the number of repetitions and the loop.

Initializing the k-means process

```
# setting up the repetitions and display options
oldpar <- par(mfrow = c(4, 4))
N = 40             # Number of repetitions
max.cluster = 15   # Maximum number of desired clusters

# initializing values
m.errs <- rep(0, max.cluster)
m.DBI <- rep(0, max.cluster)
s.errs <- rep(0, max.cluster)
s.DBI <- rep(0, max.cluster)
```

55: Although the visual inspection above suggested $k = 2$ or $k = 3$.

56: The value for the SS can be found by calling the attribute `withinss` on a `kmeans` object.

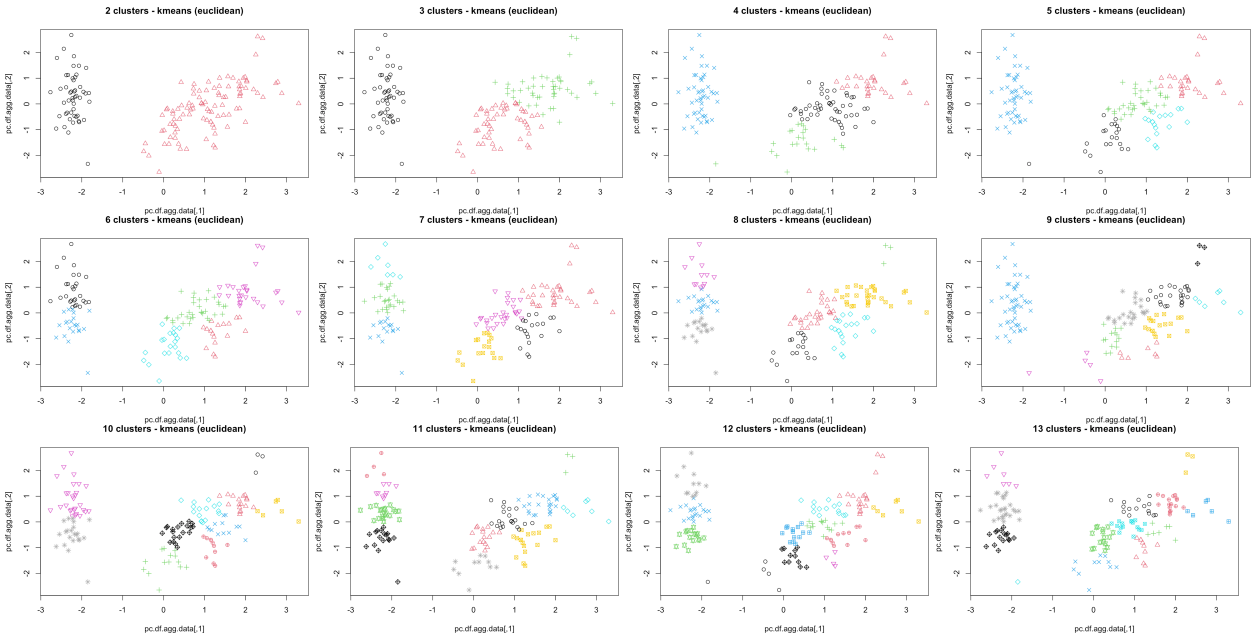
57: So that adding more clusters does not provide as big a decrease in SS.

Now we run 40 replicates for each number of clusters (so 560 calls to the `kmeans()` clustering algorithm in total). For each clustering schemes, we compute the DB index and the SS, and store them in memory.

We also print **one** of the clustering schemes for each of the number of clusters in the iteration.

Clustering the iris data

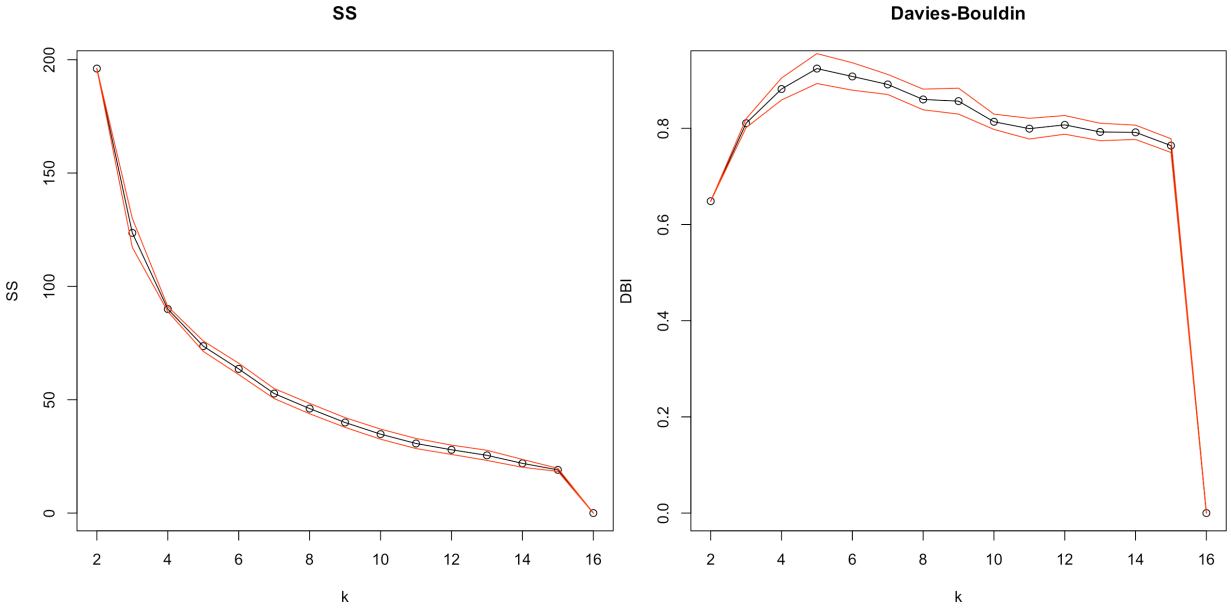
```
set.seed(0) # for replicability

## clustering and plotting
for (i in 2:max.cluster) {
  errs <- rep(0, max.cluster)
  DBI <- rep(0, max.cluster)

  for (j in 1:N) {
    KM <- kmeans(pc.df.agg.data, iter.max = 10,  i)
    errs[j] <- sum(KM$withinss)
    DBI[j] <- Davies.Bouldin(KM$centers, KM$withinss,
                             KM$size)
  }

  m.errs[i - 1] = mean(errs)
  s.errs[i - 1] = sd(errs)
  m.DBI[i - 1] = mean(DBI)
  s.DBI[i - 1] = sd(DBI)

  plot(pc.df.agg.data,col=KM$cluster, pch=KM$cluster,
       main=paste(i,"clusters - kmeans (euclidean)"))
}
```



Since we have replicates, we can compute **confidence bonds** for both the average DB index and the average SS.

Confidence bands for Within SS and DB curves

```
MSE.errs_up = m.errs + 1.96 * s.errs / sqrt(N)
MSE.errs_low = m.errs - 1.96 * s.errs / sqrt(N)

MSE.DBI_up = m.DBI + 1.96 * s.DBI / sqrt(N)
MSE.DBI_low = m.DBI - 1.96 * s.DBI / sqrt(N)

# Within SS curve
plot(2:(max.cluster+1), m.errs, main = "SS",
    xlab="k", ylab="SS")
lines(2:(max.cluster+1), m.errs)
par(col = "red")
lines(2:(max.cluster+1), MSE.errs_up)
lines(2:(max.cluster+1), MSE.errs_low)
par(col = "black")

# DBI curve
plot(2:(max.cluster+1), m.DBI, main = "Davies-Bouldin",
    xlab="k", ylab="DBI")
lines(2:(max.cluster+1), m.DBI)
par(col="red")
lines(2:(max.cluster+1), MSE.DBI_up)
lines(2:(max.cluster+1), MSE.DBI_low)
par(col = "black")
```



Where is the DB curve maximized? Does it match what the SS curve shows? We pick the optimal number of clusters using the following:

```
(i_choice <- which(
    m.DBI==max(m.DBI[1:(length(m.DBI)-1)]))+1)
```

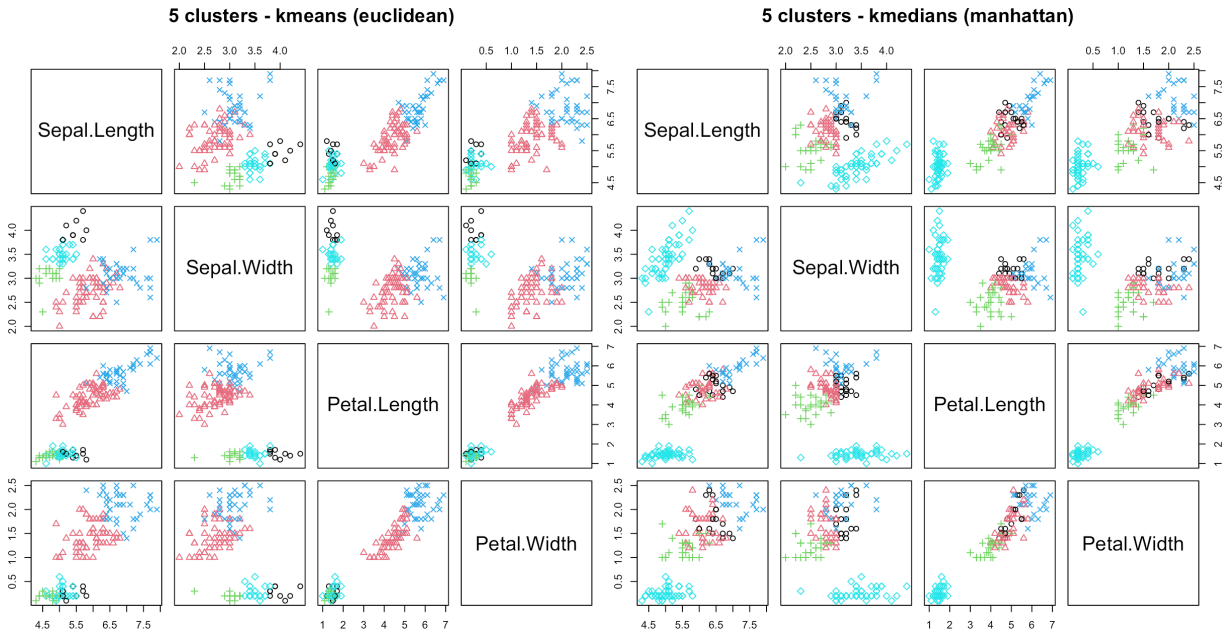
```
[1] 5
```

Finally, let us plot a "final" realization of the clustering scheme with the optimal number of clusters. We cluster on the PCA-reduced scaled data, but we plot the results with the original iris data.

We will also verify if we get similar clustering schemes when we use a different distance measure (the default measure in `kmeans()` is the Euclidean metric). Let us try the manhattan distance (*k*-medians) with the `cclus()` method (available with the `flexclust` package, which provides a more flexible clustering approach, including different algorithms and distances).

Comparison of k-means and k-medians

```
# k-means
KM <- kmeans(agg.data, iter.max = 10, i_choice)
plot(iris[,1:4],col=KM$cluster, pch=KM$cluster,
    main=paste(i_choice,"clusters - kmeans (euclidean)"))

# k-medians
library(flexclust)
KMed <- cclust(pc.df.agg.data, i_choice, dist="manhattan")
plot(iris[,1:4], col=predict(KMed), pch=predict(KMed),
     main=paste(i_choice,"clusters - kmed (manhattan)"))
```



How do they compare to one another?

19.8 Exercises

1. What are some examples of supervised and unsupervised learning tasks in the business world? In a public policy/government setting? In a scientific setting?
2. Assuming that data mining techniques are used in the following cases, identify whether the required task falls under supervised or unsupervised learning [37].

    a) Deciding whether to issue a loan to an applicant based on demographic and financial data (with reference to a database of similar data on prior customers).
    b) In an online bookstore, making recommendations to customers concerning additional items to buy based on the buying pattern in prior transactions.
    c) Identifying a network data packet as dangerous (virus, hacker attack) based on comparison to other packets with a known threat status.
    d) Identifying segments of similar customers.
    e) Predicting whether a company will go bankrupt based on comparing its financial data to those of similar bankrupt and non-bankrupt firms.
    f) Estimating the repair time required for an aircraft based on a trouble ticket.
    g) Automated sorting of mail by zip code scanning.
    h) It is more difficult and expensive to win new customers than it is to retain existing customers. Scoring each customer on their likelihood to quit can help an organization design effective interventions, such as discounts or free services, to retain profitable customers in a cost-effective manner.
    i) Some medical practitioners conduct unnecessary tests and/or over-bill their government or insurance companies. Using audit data, it may be possible to identify such providers and take appropriate action.
    j) A market basket analysis can help develop predictive models to determine which products often sell together. This knowledge of affinities between products can help retailers create promotional bundles to push non-selling items along a set of products that sell well.
    k) Diagnosing the cause of a medical condition is the crucial first step in medical engagement. In addition to the current condition, other factors can be considered, including the patient's health history, medication history, family's history, and other environmental factors. A predictive model can absorb all of the information available to date (for this patient and others) and make probabilistic diagnoses, in the form of a decision tree, taking away most of the guess work involved.
    l) Schools can develop models to identify students who are at risk of not returning to school. Such students can be flagged to be on the receiving end of potential corrective measures.
    m) In addition to customer data, telecom companies also store call detail records (CDR), which precisely describe the calling behaviour of each customer. The unique data can be used to profile customers, who may be marketed to based on the similarity of their CDR to other customers'.
    n) Statistically, all equipment is likely to break down at some point in time. Predicting which machine is likely to shut down is a complex process. Decision models to forecast machinery failure could be constructed using past data, which can lead to savings provided by preventative maintenance.
    o) Identifying which tweets contain disinformation and which tweets are legitimate.

3. Would the results of the *Danish medical study* (see Section 19.3) be applicable to the Canadian context? To the Chinese context? What do you think some of the ethical/technical challenges were?
4. Evaluate the following candidate association rules for the British Musical Dataset introduced in Section 19.3:

    a) If an individual owns a classical music album (W), then they also own a hip-hop album (Z), given that $\text{Freq}(W) = 2010$, $\text{Freq}(Z) = 6855$, and $\text{Freq}(W \cap Z) = 132$.
    b) If an individual owns both the Beatles' *Sergeant Peppers' Lonely Hearts Club Band* and a classical music album, then they were born before 1976, given that $\text{Freq}(Y \cap W) = 1852$ and $\text{Freq}(Y \cap W \cap X) = 1778$.

5. Out of the 3 rules that have been established in the previous question ($X \rightarrow Y$, $W \rightarrow Z$, and (Y AND W) $\rightarrow X$), which do you think is more useful? Which is more surprising?

6. A store that sells accessories for smart phones runs a promotion on faceplates. Customers who purchase multiple faceplates from a choice of 6 different colours get a discount. The store managers, who would like to know what colours of faceplates are likely to be purchased together, collected past transactions in the file `Transactions.csv`. Consider the following rules:

   - {**r**ed, **w**hite} → {**g**reen}
   - {**g**reen} → {**w**hite}
   - {**r**ed, **g**reen} → {**w**hite}
   - {**g**reen} → {**r**ed}
   - {**o**range} → {**r**ed}
   - {**w**hite, **b**lack} → {**y**ellow}
   - {**b**lack} → {**g**reen}

   a) For each rule, compute the support, confidence, interest, lift, and conviction.
   b) Amongst the rules for which the support is positive (> 0), which one has the highest lift? Confidence? Interest? Conviction?
   c) Build an additional 5-10 candidate rules (randomly), and evaluate them. Which of the 12-17 candidate rules do you think would be most useful for the store managers?
   d) How would one determine reasonable threshold values for the support, coverage, interest, and lift of rules derived from a given dataset?

7. Consider the following datasets:

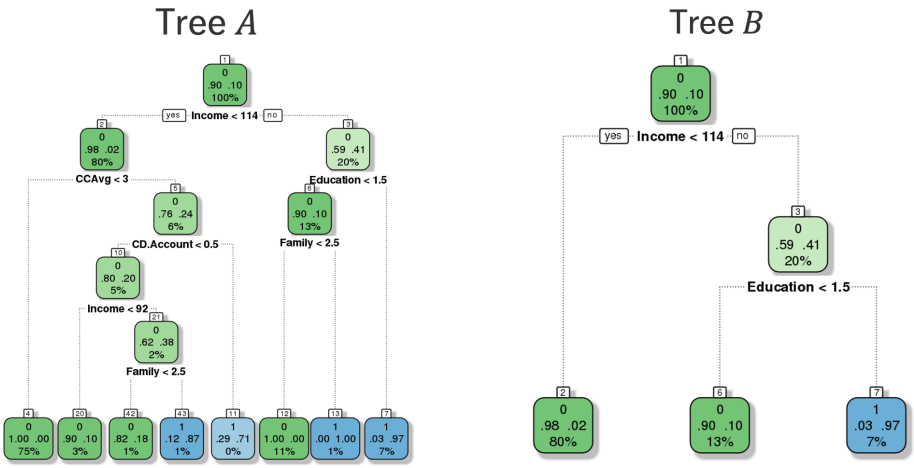
   - `GlobalCitiesPBI.csv`
   - `2016collisionsfinal.csv`
   - `polls_us_election_2016.csv`, and
   - `HR_2016_Census_simple.xlsx`.

   a) Determine what the data is reporting on / what it is about / create a "data dictionary" to explain the different fields and variables in the dataset.
   b) Develop a list of questions you would like to answer about the data.
   c) Investigate variables (individual, bivariate, multivariate) through charts, distributions, variable interactions, summary statistics, etc.
   d) Do you trust the data or not? Why? If you don't trust it, flag some potential issues with the data/specific entries.
   e) Conduct an association rule mining analysis of the datasets. Using either the brute force approach or the apriori algorithm, determine 10-20 strong association rules. Visualize them, and interpret their results.

8. *UniversalBank* is looking at converting its liability customers (i.e., customers who only have deposits at the bank) into asset customers (i.e., customers who have a loan with the bank). In a previous campaign, *UniversalBank* was able to convert 9.6% of 5000 of its liability customers into asset customers.

   The marketing department would like to understand what combination of factors make a customer more likely to accept a personal loan, in order to better design the next conversion campaign. `UniversalBank.csv` contains data on 5000 customers, including the following measurements: age, years of professional experience, yearly income (in thousands of USD), family size, value of mortgage with the bank, whether the client has a certificate of deposit with the bank, a credit card, etc.

   They build 2 decision trees on a training subset of 3000 records to predict whether a customer is likely to accept a personal loan (1) or not (0).



a) Explore the `UniversalBank.csv` dataset. Can you come up with a reasonable guess as to what each of the variables represent?
b) How many variables are used in the construction of tree A? Of tree B?
c) Are the following decision rules valid or not for trees A and/or B?

- IF (Income $\geq$ 114) AND (Education $\geq$ 1.5) THEN (Personal Loan = 1)
- IF (Income $<$ 92) AND (CCAvg $\geq$ 3) AND (CD.Account $<$ 0.5) THEN (Personal Loan = 0)

d) What prediction would trees A and B make for a customer with:

- a yearly income of 94,000$USD (Income = 94);
- 2 kids (Family = 4);
- no certificate of deposit with the bank (CD.Account = 0);
- a credit card interest rate of 3.2% (CCAvg = 3.2), and
- a graduate degree in Engineering (Education = 3)?

9. The confusion matrices for the predictions of trees A and B on the remaining 2000 testing observations are shown below.

Tree A

		Predicted A	Predicted B	Total	
Actuals	A	1792	19	1811	90.55%
	B	18	171	189	9.45%
Total		1810	190	2000	
		90.50%	9.50%		

Tree B

		Predicted A	Predicted B	Total	
Actuals	A	1801	10	1811	90.55%
	B	64	125	189	9.45%
Total		1865	135	2000	
		93.25%	6.75%		

a) Using the appropriate matrices, compute the 9 performance evaluation metrics for each of the trees (on the testing set).
b) If customers who would not accept a personal loan get irritated when offered a personal loan, what tree should *UniversalBank*'s marketing group use to help maintain good customer relations?

10. Consider the `algae_blooms.csv` dataset. We try to build a model to predict the presence/absence of algae based on various chemical properties of river water. What is the data science motivation for such a model? After all, we can simply analyze water samples to determine if various harmful algae are present or absent. The answer is simple: chemical monitoring is cheap and easy to automate, whereas biological analysis of samples is expensive and slow. Another answer is that analyzing the samples for harmful content does not provide a better understanding of algae drivers: it just tells us which samples contain algae.

a) Load the data and summarize/visualize it: you will be tasked with predicting the presence/absence of algae a1 and a2.
b) Clean the data and impute missing values, as needed.
c) Remove 20% of the observations for a validation set.

d) Create a training/testing pair on the remaining 80% of the observations and train 2 decision trees to predict the presence/absence of algaes a1 and a2, respectively. Evaluate the performance of each model. Which models performs best on your training/testing pair?
e) Repeat step d) on at least 20 distinct training/testing pairs. Evaluate the performance of each model, and save them.
f) For each algae, pick the best of the models (how would you determine this) and use it to make predictions for the readings in the validation set. Evaluate.
g) Instead of picking the best of the 20+ models, find some way to combine the results of the 20 models and to make predictions for the readings in the validation set. Evaluate these predictions.
h) Which of the resulting models of steps 6 or 7 provide the best performance? Which are easier to interpret?

11. Repeat question 10, using the same validation set in part c). In part d), use the remaining 80% of the data to build a decision tree (do not split into a training/testing pair first). Use these models to make predictions for the readings in the validation set. Evaluate these predictions. Is there evidence of overfitting?
12. Repeat question 10, using the same validation set in part c). In parts d) to g), use decision stumps (decision trees with only 1 branching point) instead of full growth trees. Is there evidence of underfitting?
13. The population of Canada is divided physically into provincial and territorial areas, most of which are further subdivided into health regions. The Census information (from 2016) is available for those health regions. The equivalent 2018 dataset has been clustered to produce peer groups: the result is shown here. The data is found in the file `HR_2016_Census_simple.xlsx`.

a) Load the data and summarize/visualize it (extract the rows with a 4-digit geocode).
b) Clean the data and impute missing values (if necessary). Scale the data and assign to a new set.
c) Run the $k-$means algorithm (with Euclidean distance) on the scaled data, using ALL the features, for $k = 3, \ldots, 16$. Use the Davies-Bouldin index and the Within-SS index to determine the optimal number of clusters. Is the optimal clustering scheme plausible?

14. Reduce the dimension of the health region dataset by running a principal component analysis (PCA) and keep the principal components that explain up to 80% of the variability in the data. Repeat step c). Are the results significantly different than they were for question 13?
15. Run $k-$means on the original health regions data (previous question) and on the reduced data, for the same range of $k-$values, but replicate the process 30+ times per value of k. What are the optimal k values in the aggregate runs?
16. Save the cluster assignments for each run with the optimal values of k found in question 13. Say that two observations A and B have similarity $w(A, B) \in [0, 1]$ if A and B lie in the same cluster in $w(A, B)$% of the runs. What are some observations with high similarity measurements? With low similarity measurements?
17. Provide a $k-$means clustering schema for the *UniversalBank* dataset.
18. The remaining exercises use the Gapminder Tools (there is also an offline version).

a) In the default configuration, we can identify some potential association rules. Using visual and ballpark estimates, evaluate the performance of the following rules:

- Income > 8000 $\rightarrow$ Life Expectancy > 70
- Income < 8000 AND Life Expectancy < 70 $\rightarrow$ Region = Africa

18.. Play around with various charts and variables and identify and evaluate 5+ additional association rules.
18.. Identify groups of similar countries, in 2018 [be sure to validate your groups using various charts].
18.. In the default configuration, follow the trajectories of Finland, Sweden, Iceland, Norway, and Denmark between 1900 and 2018. Do the countries appear to follow similar trajectories? Are there outliers or anomalous trajectories?
18.. Repeat step d) for Brazil, Paraguay, Uruguay, Venezuela, Colombia, Peru, and Ecuador.
18.. Based on your results in steps 4 and 5, would you expect the trajectory for Argentina to be more like those of the Nordic countries or those of the South American countries? Or perhaps neither? Is your answer the same over all time horizons?

Chapter References

[1] C.C. Aggarwal. *Data Mining: the Textbook*. Cham: Springer, 2015.

[2] C.C. Aggarwal, ed. *Data Classification: Algorithms and Applications*. CRC Press, 2015.

[3] C.C. Aggarwal and C.K. Reddy, eds. *Data Clustering: Algorithms and Applications*. CRC Press, 2014.

[4] C.C. Aggarwal and P.S. Yu. 'A New Framework for Itemset Generation'. In: *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. PODS '98. Seattle, Washington, USA: Association for Computing Machinery, 1998, pp. 18–24. DOI: 10.1145/275487.275490.

[5] F.R. Bach and M.I. Jordan. 'Learning Spectral Clustering, With Application To Speech Separation'. In: *J. Mach. Learn. Res.* 7 (Dec. 2006), pp. 1963–2001.

[6] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge Press, 2012.

[7] S.E. Brossette et al. 'Association Rules and Data Mining in Hospital Infection Control and Public Health Surveillance'. In: *Journal of the American Medical Informatics Association* 5.4 (July 1998), pp. 373–381. DOI: 10.1136/jamia.1998.0050373.

[8] Subaru Canada. *Athlete Rebate*.

[9] J. Chambers and T. Hastie. *Statistical Models in S*. Wadsworth and Brooks/Cole, 1992.

[10] Z. Cheng et al. 'Exploring Millions of Footprints in Location sharing services'. In: *ICWSM*. Ed. by Lada A. Adamic, Ricardo Baeza-Yates, and Scott Counts. The AAAI Press, 2011.

[11] T. Chou. 'Apriori: Association Rule Mining In-Depth Explanation and Python Implementation'. In: *Towards Data Science* (Oct. 2020).

[12] J. Cranshaw et al. 'The Livehoods Project: Utilizing Social Media to Understand the Dynamics of a City'. In: *ICWSM*. Ed. by John G. Breslin et al. The AAAI Press, 2012.

[13] J. d'Huy. 'Scientists Trace Society's Myths to Primordial Origins'. In: *Scientific American (Online)* (Sept. 2016).

[14] B. Desgraupes. *clusterCrit: Clustering Indices*. R package version 1.2.8. 2018.

[15] D. Dua and E. Karra Taniskidou. *UCI Machine Learning Repository*. 2017.

[16] B. Efron. *Large Scale Inference: Empirical Bayes Methods for Estimation, Testing, and Prediction*. Cambridge University Press, 2010.

[17] V. Estivill-Castro. 'Why so Many Clustering Algorithms: A Position Paper'. In: *SIGKDD Explor. Newsl.* 4.1 (June 2002), pp. 65–75.

[18] S. Fefilatyev et al. 'Detection of Anomalous Particles from Deepwater Horizon Oil Spill Using SIPPER3 Underwater Imaging Platform'. In: *Data Mining Case Studies IV, Proceedings of the 11th IEEE International Conference on Data Mining*. Vancouver, BC: IEEE, 2011.

[19] R. A. Fisher. 'The Use of Multiple Measurements in Taxonomic Problems'. In: *Annals of Eugenics* 7.7 (1936), pp. 179–188.

[20] E. Garcia et al. 'Drawbacks and solutions of applying association rule mining in learning management systems'. In: *Proceedings of the International Workshop on Applying Data Mining in e-Learning*. Greece: CEUR-WS.org, 2007.

[21] U. Habib, K. Hayat, and G. Zucker. 'Complex building's energy system operation patterns analysis using bag of words representation with hierarchical clustering'. In: *Complex Adapt. Syst. Model.* 4 (2016), p. 8. DOI: 10.1186/s40294-016-0020-0.

[22] M. Hahsler and K. Hornik. 'New probabilistic interest measures for association rules'. In: *CoRR* abs/0803.0966 (2008).

[23] N. Harris. *Visualizing DBSCAN Clustering*.

[24] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, *2nd ed.* Springer, 2008.

[25] T. Hastie, T. Tibshirani, and M. Wainwright. *Statistical Learning with Sparsity: The LASSO and Generalizations*. CRC Press, 2015.

[26] K.-W. Hsu et al. 'Data Mining Based Tax Audit Selection: A Case Study of a Pilot Project at the Minnesota Department of Revenue'. In: *Real World Data Mining Applications*. Cham: Springer International Publishing, 2015, pp. 221–245. DOI: 10.1007/978-3-319-07812-0_12.

[27] G. James et al. *An Introduction to Statistical Learning: With Applications in R*. Springer, 2014.

[28] A. Jawad, K. Kersting, and N. Andrienko. 'Where Traffic Meets DNA: Mobility Mining Using Biological Sequence Analysis Revisited'. In: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS '11. Chicago, Illinois: Association for Computing Machinery, 2011, pp. 357–360. DOI: 10.1145/2093973.2094022.

[29] A.B. Jensen et al. 'Temporal disease trajectories condensed from population-wide registry data covering 6.2 million patients'. English. In: *Nature Communications* 5 (2014). DOI: 10.1038/ncomms5022.

[30] M.R. Keeler. *Nothing to Hide: Privacy in the 21st Century*. iUniverse, 2006.

[31] B. Kitts. 'Product Targeting From Rare Events: Five Years of One-to-One Marketing at CPI'. In: *Marketing Science Conference* (2005).

[32] B. Kitts. 'The Making of a Large-Scale Ad Server'. In: *Data Mining Case Studies Workshop and Practice Prize 5.* Proceedings of the IEEE Thirteenth International Conference on Data Mining Workshops. Dallas, TX: IEEE Press, 2013.

[33] B. Kitts et al. 'Click Fraud Detection: Adversarial Pattern Recognition over 5 Years at Microsoft'. In: *Annals of Information Systems (Special Issue on Data Mining in Real-World Applications)*. Springer, 2015, pp. 181–201. DOI: 10.1007/978-3-319-07812-0.

[34] H. T. Kung and D. Vlah. 'A Spectral Clustering Approach to Validating Sensors via Their Peers in Distributed Sensor Networks'. In: *Int. J. Sen. Netw.* 8.3/4 (Oct. 2010), pp. 202–208. DOI: 10.1504/IJSNET.2010.036195.

[35] O. Leduc and P. Boily. 'Boosting with AdaBoost and Gradient Boosting'. In: *Data Action Lab Blog* (2019).

[36] J. Leskovec, A. Rajamaran, and J.D. Ullman. *Mining of Massive Datasets*. Cambridge Press, 2014.

[37] A.K. Maheshwari. *Business Intelligence and Data Mining*. Business Expert Press, 2015.

[38] Q.E. McCallum. *Bad Data Handbook*. O'Reilly, 2013.

[39] A. Ng and K. Soo, eds. *Surviving a Disaster, in Numsense!* algobeans, 2016.

[40] E.R. Omiecinski. 'Alternative interest measures for mining associations in databases'. In: *IEEE Transactions on Knowledge and Data Engineering* 15.1 (2003), pp. 57–69. DOI: 10.1109/TKDE.2003.1161582.

[41] M. Orlowska et al. 'A Comparison of Antioxidant, Antibacterial, and Anticancer Activity of the Selected Thyme Species by Means of Hierarchical Clustering and Principal Component Analysis'. In: *Acta Chromatographica Acta Chromatographica* 28.2 (2016), pp. 207–221. DOI: 10.1556/achrom.28.2016.2.7.

[42] V. U. Panchami and N. Radhika. 'A novel approach for predicting the length of hospital stay with DBSCAN and supervised classification algorithms'. In: *ICADIWT*. IEEE, 2014, pp. 207–212.

[43] V. U. Panchami and N. Radhika. 'A novel approach for predicting the length of hospital stay with DBSCAN and supervised classification algorithms'. In: *ICADIWT*. IEEE, 2014, pp. 207–212.

[44] G. Piatetsky-Shapiro. 'Discovery, Analysis, and Presentation of Strong Rules'. In: *Knowledge Discovery in Databases*. 1991.

[45] C. Plant et al. 'Automated detection of brain atrophy patterns based on MRI for the prediction of Alzheimer's disease'. In: *NeuroImage* 50.1 (2010), pp. 162–174.

[46] F. Provost and T. Fawcett. *Data Science for Business*. O'Reilly, 2015.

[47] A.M. Raja. 'Penguins Dataset Overview - iris alternative'. In: *Towards Data Science* (June 2020).

[48] M. Risdal. 'Exploring Survival on the Titanic'. In: *Kaggle.com* (2016).

[49] C.F. Robert. *Le choix bayésien - Principes et pratique*. Springer-Verlag France, 2006.

[50] D. Robinson. 'What's the difference between data science, machine learning, and artificial intelligence? ↗'. In: *Variance Explained* (Jan. 2018).

[51] G. Schoier and G. Borruso. 'Individual Movements and Geographical Data Mining. Clustering Algorithms for Highlighting Hotspots in Personal Navigation Routes'. In: *Computational Science and Its Applications - ICCSA 2011*. Ed. by Beniamino Murgante et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 454–465.

[52] E. Schubert et al. 'DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN'. In: *ACM Trans. Database Syst.* 42.3 (July 2017). DOI: 10.1145/3068335.

[53] E. Siegel. *Predictive Analytics: The Power to Predict Who Will Click, Buy, Lie or Die*. Predictive Analytics World, 2016.

[54] P.-N. Tan, V. Kumar, and J. Srivastava. 'Selecting the Right Objective Measure for Association Analysis'. In: *Inf. Syst.* 29.4 (June 2004), pp. 293–313. DOI: 10.1016/S0306-4379(03)00072-3.

[55] L. Torgo. *Data Mining with R, 2nd ed.* CRC Press, 2016.

[56] Wikipedia. *Association Rule Learning* ↗. 2020.

[57] Wikipedia. *Cluster Analysis Algorithms* ↗.

[58] D.H. Wolpert. 'The Lack of A Priori Distinctions Between Learning Algorithms'. In: *Neural Computation* 8.7 (1996), pp. 1341–1390. DOI: 10.1162/neco.1996.8.7.1341.

[59] D.H. Wolpert and W.G. Macready. 'Coevolutionary free lunches'. In: *IEEE Transactions on Evolutionary Computation* 9.6 (2005), pp. 721–735. DOI: 10.1109/TEVC.2005.856205.

[60] D. Woods. 'bitly's Hilary Mason on "What is A Data Scientist?" ↗'. In: *Forbes* (Mar. 2012).

Regression and Value Estimation 20

by **Patrick Boily**

In Chapter 19 (*Machine Learning 101*), we provided a (basically) math-free general overview of machine learning. In this chapter, we present an introductory mathematical treatment of the discipline, with a focus on regression and value estimation methods (in particular, on parametric methods).

Our approach borrows heavily from [3, 5]; explanations and examples are also available in [1].

We will continue the ML treatment in Chapters 21 and 22 (and 23, to a lesser extent).

20.1 Statistical Learning

Statistical learning is a series of procedures and approaches that allows analysts to tackle problems such as:

- identifying risk factors associated to breast/prostate cancer;
- predicting whether a patient will have a second, fatal heart attack within 30 days of the first on the basis of demographics, diet, clinical measurements, etc.;
- establishing the relationship between salary and demographic information in population survey data;
- predicting the yearly inflation rate using various indicators, etc.

Statistical learning tasks are typically divided into 2 main classes: **supervised learning** and **unsupervised learning**.[1]

1: There are other types, such as semi-supervised or reinforcement learning, but these are topics for future chapters.

20.1.1 Supervised Learning Framework

In the supervised learning environment, the **outcome** (response, target, dependent variable, etc.) is denoted by Y, and the vector of p **predictors** (features) by $\vec{X} = (X_1, \ldots, X_p)$.

If Y is quantitative (price, height, etc.), then the problem of predicting Y in terms of $\vec{X}$ is a **regression** task; if Y takes on values in a finite unordered set (survived/died, colours, vegetation types, etc.), it is a **classification** task. This is typically achieved with the use of **training data**, which is to say historical observations or instances, which we often denote by $[\mathbf{X} \mid \mathbf{Y}]$ (the column denoting the observation IDs is dropped).

obs.	predictors	predictors	predictors	resp.
1	$x_{1,1}$	$\cdots$	$x_{1,p-1}$	y_1
$\vdots$	$\vdots$		$\vdots$	$\vdots$
n	$x_{n,1}$	$\cdots$	$x_{n,p-1}$	y_n

The objectives of supervised learning are usually to:

- accurately predict **unseen** test cases;
- understand which inputs affect the outcomes (if any), and how;
- assess the quality of predictions and/or inferences made on the basis of the training data, etc.

In unsupervised learning, on the contrary, there are no outcome variables, only features on a set of observations $\mathbf{X}$.[2]

2: The response variable $\mathbf{Y}$ that was segregated away from $\mathbf{X}$ in the supervised learning case could now be one of the variables in $\mathbf{X}$.

The objectives are much more **vague** – analysts could seek to:

- find sets of features that behave similarly across observations;
- find combinations of features with the most variation;
- find natural groups of similar observations, etc.

We will discuss such methods in detail in Chapter 22.

Statistical Learning vs. Machine Learning The term "statistical learning" is not used frequently in practice;[3] we speak instead of **machine learning**. If a distinction must be made, we could argue that:

3: Except by mathematicians and statisticians, perhaps.

- statistical learning arises from statistical-like models, and the emphasis is usually placed on **interpretability**, **precision**, and **uncertainty**, whereas
- machine learning arise from artificial intelligence studies, with emphasis on **large scale applications** and **prediction accuracy**.

The dividing line between the terms is blurry – the vocabulary used by practitioners mostly betrays their educational backgrounds (but see [7] for another take on this).

Motivating Example Throughout, we will illustrate the concepts and notions *via* the `gapminder.csv` dataset, which records socio-demographic information relating to the planet's nations, from 1960 to 2011 [9, 8].

We will be interested in determining if there is a link between life expectancy, at various moments in time, and the rest of the predictors.

The dataset contains 7139 observations of 9 columns:

- a country × year identifier (2 variables, i and X_1);
- a region and continent pair of categorical predictors (2 variables, X_2 and X_3);
- four numerical predictors: population X_4, infant mortality X_5, fertility X_6, gross domestic product in 1999 dollars X_7, and
- life expectancy Y, the numerical response.

Setting up the Gapminder dataset

```
library(dplyr)
gapminder.ML = read.csv("gapminder.csv",
                        stringsAsFactors=TRUE)
gapminder.ML <- gapminder.ML[complete.cases(gapminder.ML),]
gapminder.ML <- gapminder.ML[,c("country","year","region",
        "continent","population","infant_mortality",
        "fertility","gdp","life_expectancy")]
```

The structure is provided below:

```
str(gapminder.ML)
```

```
'data.frame':   7139 obs. of  9 variables:
 $ country         : Factor w/ 185 levels "Albania","Algeria",..: 2 5 8 9 11 13 14 16 18 20 ...
 $ year            : int  1960 1960 1960 1960 1960 1960 1960 1960 1960 1960 ...
 $ region          : Factor w/ 22 levels "Australia and New Zealand",..: 11 15 1 22 2 18 2 ...
 $ continent       : Factor w/ 5 levels "Africa","Americas",..: 1 2 5 4 2 3 2 4 1 2 ...
 $ population      : int  11124892 20619075 10292328 ...
 $ infant_mortality: num  148.2 59.9 20.3 37.3 51 ...
 $ fertility       : num  7.65 3.11 3.45 2.7 4.5 6.73 4.33 2.6 6.28 6.7 ...
 $ gdp             : num  1.38e+10 1.08e+11 9.67e+10 5.24e+10 1.31e+09 ...
 $ life_expectancy : num  47.5 65.4 70.9 68.8 62 ...
```

In other words, we will be looking for **models** of the form

$$Y = f(X_1, \ldots, X_7) + \varepsilon \equiv f(\vec{X}) + \varepsilon,$$

where f is the **systematic component of** Y **explained by** X, and ε is the **random error term**, which accounts for measurement errors and other deviations and discrepancies.[4]

4: Generally, we require $\mathrm{E}(\varepsilon) = 0$.

20.1.2 Systematic Component and Regression

It is the systematic component that is used for **predictions** and **inferences**. As long as f is "good", we can:

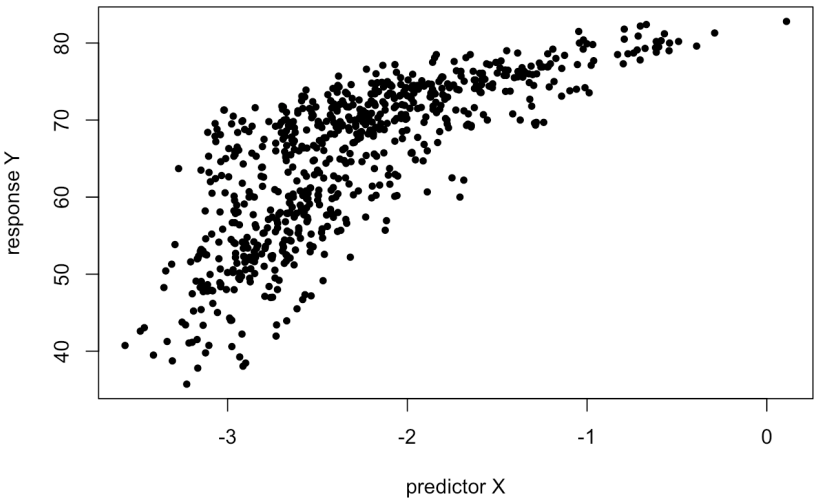
- make predictions for the response Y at new points $\vec{X} = \mathbf{x}$;
- understand which features of $\vec{X} = (X_1, \ldots, X_p)$ are important to explain the variation in Y, and
- depending on the complexity of f, understand the effect of each feature X_j on Y.

Imagine a model with one predictor X and a target Y, with systematic component f, so that

$$Y = f(X) + \varepsilon.$$

For instance, consider the following subset of the Gapminder dataset.

```
attach(gapminder.ML)
x=log(fertility[10*(1:730)]/infant_mortality[10*(1:730)])
y=life_expectancy[10*(1:730)]
x=x[!is.na(x)]
y=y[!is.na(y)]
plot(x,y,ylab="response Y", xlab="predictor X", pch=20)
```



What is the ideal f in this case? How can we find it?

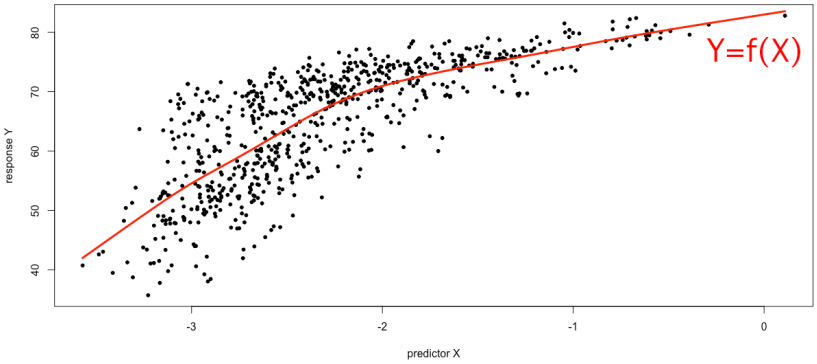


Figure 20.1: Regression model for a subset of the Gapminder data.

In that case, what would be a good value of $f(-2)$, say?

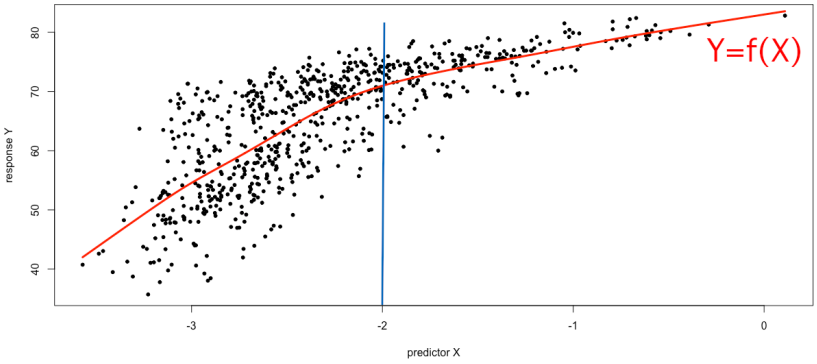


Figure 20.2: Regression model for a subset of the Gapminder data, with vertical line at $X = -2$.

Ideally, we would like to have $f(-2) = \mathrm{E}[Y \mid X = -2]$.[5] For any x in the range of X, the function

$$f(x) = \mathrm{E}[Y \mid X = x]$$

5: Why?

is the **regression function of** Y **on** X. In the general setting with p predictors, the regression function is

$$f(\mathbf{x}) = f(x_1, \ldots, x_p) = \mathrm{E}[Y \mid X_1 = x_1, \ldots, X_p = x_p] = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}].$$

It is **optimal** in the sense that this regression function minimizes the average square deviation from the response variable, that is to say,

$$f = \arg\min_g \left\{ \mathrm{E}\left[(Y - g(\vec{X}))^2 \mid \vec{X} = \mathbf{x} \right] \right\}.$$

The term

$$\varepsilon = \varepsilon_{\vec{X}} = Y - f(\vec{X})$$

is the **irreducible error** of the regression. Typically, $\varepsilon_{\vec{X}} \neq 0$ for all $\vec{X}$, since, even when f is known exactly, there will still be some uncertainty in the predictions due to some noise-generating mechanism in the "real world".

If $\hat{f}$ is any estimate of the regression function f,[6] then

$$\begin{aligned} \mathrm{E}[(Y - \hat{Y})^2 \mid \vec{X} = \mathbf{x}] &= \mathrm{E}[(f(\vec{X}) + \varepsilon - \hat{f}(\vec{X}))^2 \mid \vec{X} = \mathbf{x}] \\ &= \underbrace{[f(\mathbf{x}) - \hat{f}(\mathbf{x})]^2}_{\text{reducible}} + \underbrace{\mathrm{Var}(\varepsilon)}_{\text{irreducible}} . \end{aligned}$$

Since the **irreducible component** is not a property of the estimate $\hat{f}$, the objective of minimizing $\mathrm{E}[(Y - \hat{Y})^2]$ can only be achieved by working through the **reducible component**. When we speak of **learning** a model, we mean that we use the training data to find an estimate $\hat{f}$ of f that minimizes this reducible component, in some way.

6: In particular, $\hat{f}(\vec{X}) = \hat{Y} \approx Y = f(\vec{X}) + \varepsilon$.

Estimating the Regression Function In theory, we know that the regression function is

$$f(\mathbf{x}) = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}];$$

in practice, however, there might be too few (or even no) observations at $\vec{X} = \mathbf{x}$ to trust the estimate provided by the sample mean. One solution is to approximate the expectation by a **nearest neighbour average**

$$\hat{f}(\mathbf{x}) = \mathrm{Avg}\{Y \mid \vec{X} \in N(\mathbf{x})\},$$

where $N(\mathbf{x})$ is a neighbourhood of $\mathbf{x}$.

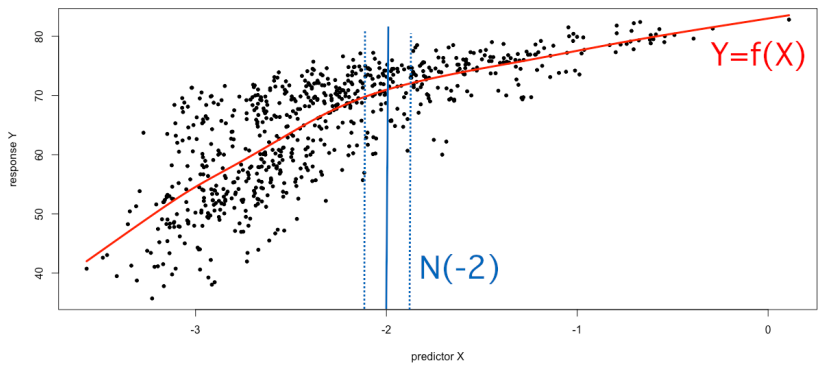


Figure 20.3: Regression model for a subset of the Gapminder data, with vertical line at $X = -2$ and neighbourhood $N(-2)$.

In general, this approach works reasonably well when p is "small" ($p \leq 4$?) and N is "large", but it fails when p is too large because of the **curse of dimensionality**. The problem is that nearest neighbours are usually far when p is large. Indeed, if $N(\mathbf{x})$ is defined as the nearest 5% of observations to $\mathbf{x}$, say,[7] then we need to leave the "local" neighbourhood of $\mathbf{x}$ to build $N(\mathbf{x})$, which could compromise the quality of $\hat{f}(\mathbf{x})$ as an approximation to $f(\mathbf{x})$.

7: The proportion must be large enough to bring the variance down.

We provide more details in Chapter 23, but this is a topic about which it is worth being well-read (see [3] for a formal treatment).

The various statistical learning methods attempt to provide estimates of the regression function by minimizing the reducible component through **parametric** or **non-parametric** approaches.[8] For instance, the **classical linear regression** approach is parametric: it assumes that the true regression function f is linear and suggests the estimate

$$f_L(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p.$$

8: In this context, "parametric" means that assumptions are made about the form of the regression function f; "non-parametric" means that no such assumptions are made.

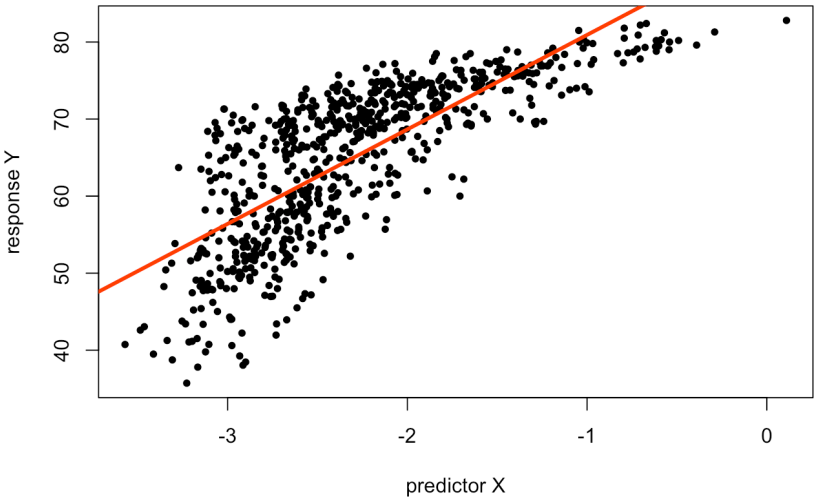
The objective, in this case, is to learn the $p+1$ parameters $\beta_0, \beta_1, \ldots, \beta_p$ with the help of the training data.

In practice, this assumption almost never holds, but it often provides an **interpretable**[9] approximation to the true regression function f (see below for an example).

9: We will revisit this concept at a later stage.

Gapminder subset and linear regression

```
lin.reg = lm(y~x)
plot(x,y,ylab="response Y", xlab="predictor X", pch=20)
abline(lin.reg, col="red", lwd=3)
```



As an example, if the true fit of the motivating example was

$$\text{life expectancy} = f(\text{fertility, infant mortality, gdp}) + \varepsilon,$$

say, then the linear regression approach would assume that

$$\begin{aligned} f(\text{fertility,infant mortality, gdp}) &\approx f_L(\text{fertility, infant mortality, gdp}) \\ &= \beta_0 + \beta_1 \cdot \text{fertility} + \beta_2 \cdot \text{infant mortality} + \beta_3 \cdot \text{gdp}. \end{aligned}$$

The main advantages of the linear model are that it is interpretable and that it is easier to learn $p + 1$ parameters than it is to learn a whole function f. On the flip side, the linear model does not usually match the true regression function f; if $f_L \not\approx f$, then predictions will suffer.

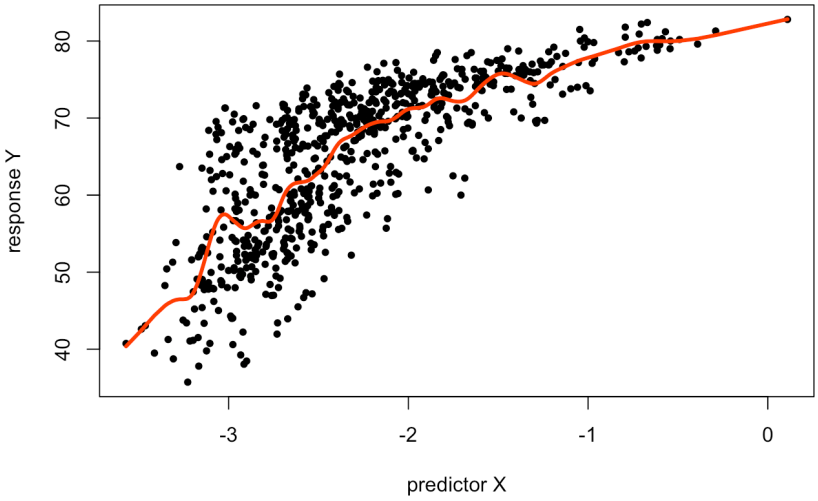
We could decide to consider more complex functions in order to get better estimates (and thus better prediction accuracy), but this comes at a **cost** – the resulting functions are usually more difficult to learn and they tend to **overfit the data**.[10]

10: Which is to say, they mistake noise in the data for a signal to model, see Section 20.1.4 for details.

Splines provide examples of non-parametric models (see Section 20.5.2): they make no assumption about the form of f – they simply seek to estimate f by getting close to the data points without being too **rough** or too **wiggly**, as below.

Gapminder subset and smoothing spline

```
smoothingSpline = smooth.spline(x, y, spar=0.7)
plot(x,y,ylab="response Y", xlab="predictor X", pch=20)
lines(smoothingSpline, col="red", lwd=3)
detach(gapminder.ML)
```



The main advantage of non-parametric approaches is that they have the potential to fit a **wider range** of regression shapes. But since estimating f is not reduced to learning a small number of parameters, substantially more data is required to obtain accurate estimates.[11]

11: And the whole situation is susceptible to overfitting.

Non-parametric methods are usually more **flexible** (they can produce a large range of shapes when estimating the true regression function f); parametric models are usually more **interpretable**.[12]

12: The set of parameters to learn is small and we can more easily make sense of them, which leads us to a better understanding of how the predictors interact to produce outputs.

Approaches that provide:

- high flexibility, but low interpretability include ensemble learning, support vector machines, neural networks, and splines;
- low flexibility, but high interpretability include the LASSO and OLS, and
- medium flexibility and medium interpretability include generalized additive models and regression trees.

There are no **high-flexibility/high-interpretability** approaches. The **trade-off** between two competing desirable properties is the calling card of machine learning; we will encounter such trade-offs time and time again; they dictate what the discipline can and cannot hope to achieve.

20.1.3 Model Evaluation

13: From a model performance point of view.

In an ideal world,[13] we would want to identify the modeling approach that performs "best", and use it for all problems.

The discussion on **trade-offs** shows that the concept of "best performance" is impossible to define in practice in a way that meets all desired requirements, and a balance must be struck. Another issue lurks around the corner, even when we settle on an "optimal" performance evaluation measure: no single method is optimal over all possible datasets.[14]

14: In reality, machine learning is simply applied optimization; the proof of this **No-Free Lunch Theorem** falls outside the scope of this document (but see [13, 12] for details).

Given a specific task and dataset, then, how do we select the approach that will yield the best results (for a given value of "best")? In practice, **this is the main machine learning challenge.**

In order to evaluate a model's performance at a specific task, we must be able to measure how well predictions match the **observed data**. In a **regression/value estimation setting**, various metrics are used:

- **mean squared error** (MSE):

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{f}(\mathbf{x}_i))^2;$$

- **mean absolute error** (MAE):

$$\frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{f}(\mathbf{x}_i)|;$$

- **normalized mean squared error** (NMSE):

$$\frac{\sum_{i=1}^{N}(y_i - \hat{f}(\mathbf{x}_i))^2}{\sum_{i=1}^{N}(y_i - \overline{y})^2};$$

- **normalized mean absolute error** (NMAE):

$$\frac{\sum_{i=1}^{N}|y_i - \hat{f}(\mathbf{x}_i)|}{\sum_{i=1}^{N}|y_i - \overline{y}|};$$

- **mean average percentage error** (MAPE):

$$\frac{1}{N}\sum_{i=1}^{N}\frac{|y_i - \hat{f}(\mathbf{x}_i)|}{y_i};$$

- **correlation** $\rho_{\hat{y},y}$, etc.

The MSE has convenient mathematical properties, and we will follow the lead of just about every reference in making it our go-to metric, but note that the conceptual notions we will discuss would be qualitatively similar for all performance evaluation tools.

Note that in order to evaluate the suitability of a model for **predictive** purposes, these metrics should be evaluated on **testing data** (or unseen data), not on the training data.[15]

15: Failure to do so means that the model can at best be used to describe the training dataset (which might still be a valuable contribution).

For instance, if we are trying to determine whether any clinical measurement in patients are likely to predict the onset of Alzheimer's disease, we do not particularly care if the algorithm does a good job of telling us that the patients we have already tested for the disease have it or not – it is new patients that are of interest.[16]

16: Although it would be surprising if the performance on the test data is any good if the performance on the training data is middling. We shall see at a later stage that the training/testing paradigm can also help with problems related to overfitting.

Let $\text{Tr} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, N\}$ be the **training set** and suppose that we use some statistical learning method to estimate the true relationship $Y = f(\vec{X}) + \varepsilon$ by $\hat{Y} = \hat{f}(\vec{X})$, i.e., **we fit $\hat{f}$ over Tr**.

Hopefully, we have $\hat{f}(\mathbf{x}_i) \approx y_i$ for all $i = 1, \ldots, N$, and

$$\text{MSE}_{\text{Tr}} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{f}(\mathbf{x}_i))^2$$

is small.

If it is indeed small, then the model does a good job of **describing** Tr. But, as discussed above, this is largely irrelevant to (if not uncorrelated with) our ability to make **good predictions**; what we would really like to know is if

$$\hat{f}(\mathbf{x}^*) \approx f(\mathbf{x}^*) = y^*$$

for observations $(\mathbf{x}^*, y^*) \notin \text{Tr}$.

An **optimal** statistical learning method for a given combination of task and dataset is one that minimizes

$$\text{MSE}_{\text{Te}} = \frac{1}{M} \sum_{j=N+1}^{N+M=n} (y_j - \hat{f}(\mathbf{x}_j))^2$$

over the testing set $\text{Te} = \{(\mathbf{x}_j, y_j) \mid j = N+1, \ldots, N+M = n\}$, where, *a priori*, none of the test observations were in Tr.[17] The general situation is illustrated in Figures 20.4 and 20.5.

17: New test observations may end up assuming the same values as some of the training observations, but that is an accident of sampling and/or it is due to the reality of the scenario under consideration.

20.1.4 Bias-Variance Trade-Off

The "U" shape of the testing MSE in Figure 20.5 is **generic** – something of this nature occurs for nearly all datasets and choice of supervised learning family of methods (for regression and for classification): **underfitting** and **overfitting** is a fact of machine learning life.

The generic shape can be explained by two properties of SL methods: the **bias** and the **variance**. Consider a test observation $(\mathbf{x}^*, y^*)$, and a fitted model $\hat{f}$ (trained on Tr), which approximates the true model

$$Y = f(\vec{X}) + \varepsilon, \quad \text{where } f(\mathbf{x}) = \text{E}[Y \mid \vec{X} = \mathbf{x}].$$

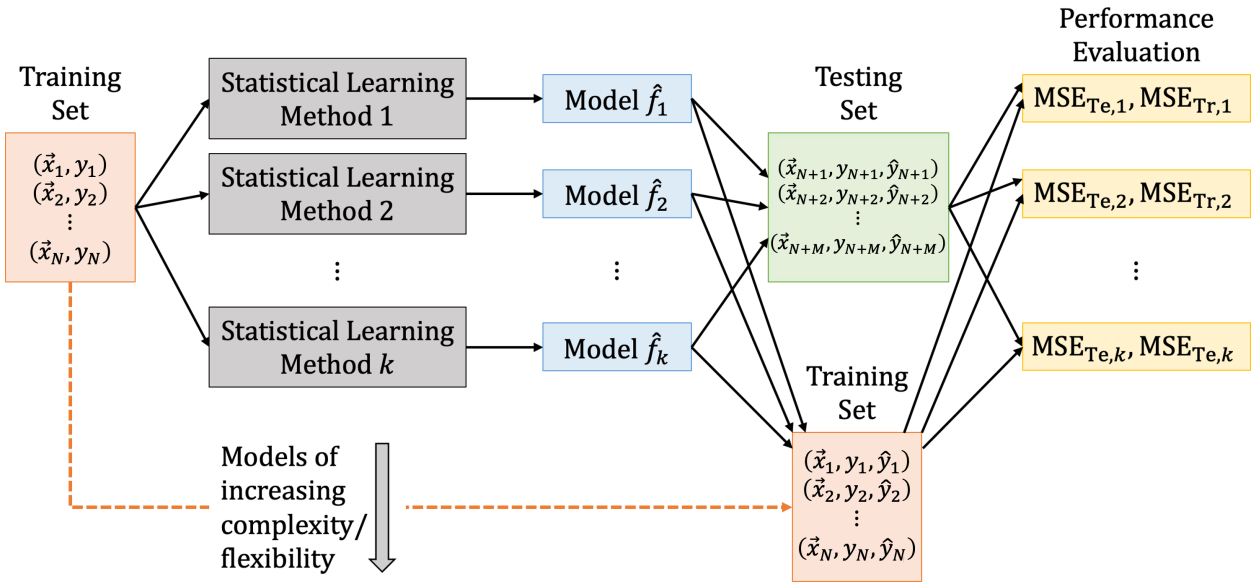


Figure 20.4: The training/testing paradigm. Training data is fed into a variety of statistical learning methods, possibly arranged in increasing order of complexity, yielding a sequence of models. These models are then used to make predictions on the testing set (using only the predictors variables); the predictions are then compared with the actual values to evaluate the performance of the models on the testing set. The performance of the models on the training set can also be evaluated.

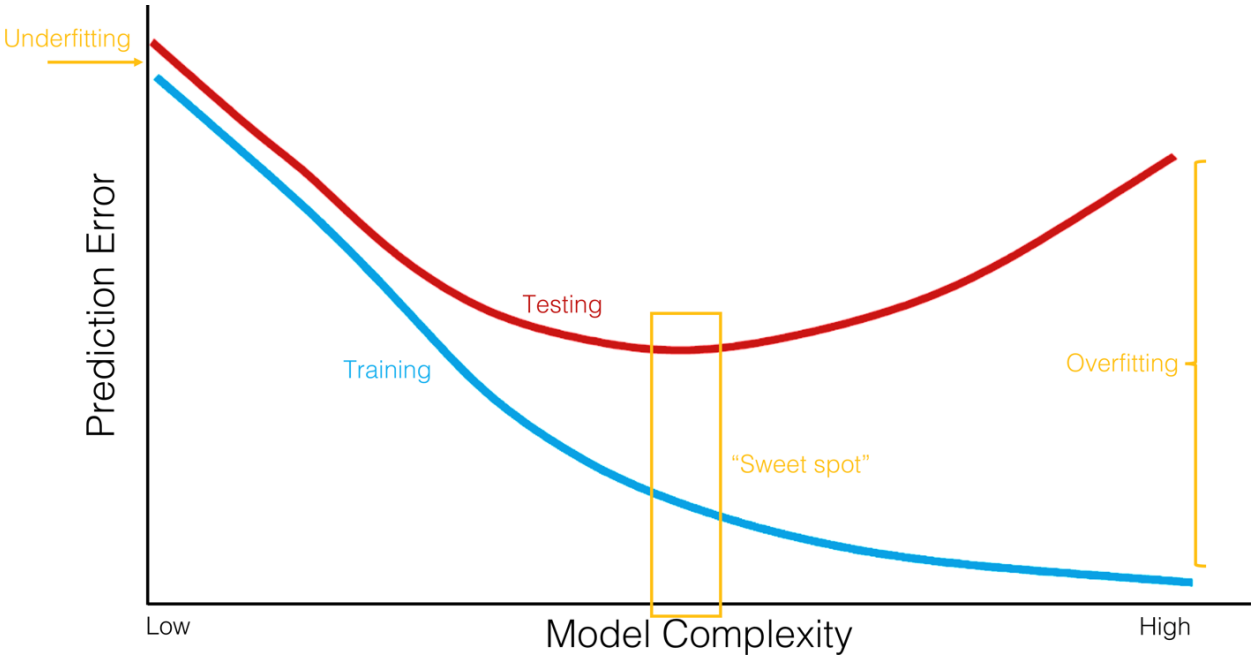


Figure 20.5: Generic illustration of the bias-variance trade-off; when the complexity of the model increases, the training error decreases, but the testing error eventually starts increasing. Generally, models that are too simple will have 'large' prediction errors on both the training and the testing sets (underfitting), whereas for models that are too complex, the training error tends to be "small" while the testing error tends to be "large" (overfitting). Based on [5, 3].

The **expected test MSE at** $\mathbf{x}^*$ can be decomposed into 3 fundamental quantities

$$\begin{aligned}\mathrm{E}\left[\mathrm{MSE}_{\mathrm{Te}}(\mathbf{x}^*)\right] &= \mathrm{E}\left[(y^* - \hat{f}(\mathbf{x}^*))^2\right] \\ &= \underbrace{\mathrm{Var}(\hat{f}(\mathbf{x}^*))}_{\text{variance}} + \underbrace{\left\{\mathrm{E}\left[\hat{f}(\mathbf{x}^*) - f(\mathbf{x}^*)\right]\right\}^2}_{\text{squared bias}} + \mathrm{Var}(\varepsilon).\end{aligned}$$

As before, $\mathrm{Var}(\varepsilon)$ is the **irreducible error** (due to the inherent noise in the data); the **variance component error** $\mathrm{Var}(\hat{f}(\mathbf{x}^*))$ arises since different training sets would yield different fitted models $\hat{f}$, and the **(squared) bias component error** arises, in part, due to the "difficult" problem being approximated by a "simple" model (see [5, 3] for details).

The **overall expected test MSE** $\mathrm{E}[\mathrm{MSE}_{\mathrm{Te}}]$ is the average of $\mathrm{E}[\mathrm{MSE}_{\mathrm{Te}}(\mathbf{x}^*)]$ over all allowable $\mathbf{x}^*$ in the testing space. Note that

$$\mathrm{E}[\mathrm{MSE}_{\mathrm{Te}}] \geq \mathrm{Var}(\varepsilon),$$

by construction.

In general, more flexible methods (i.e., more complex methods) tend to have higher variance and lower bias, and *vice-versa*: simpler methods have higher bias and lower variance. It is this interplay between bias and variance that causes models to underfit (high bias) or overfit (high variance) the data (see **bias-variance trade-off** diagram below).

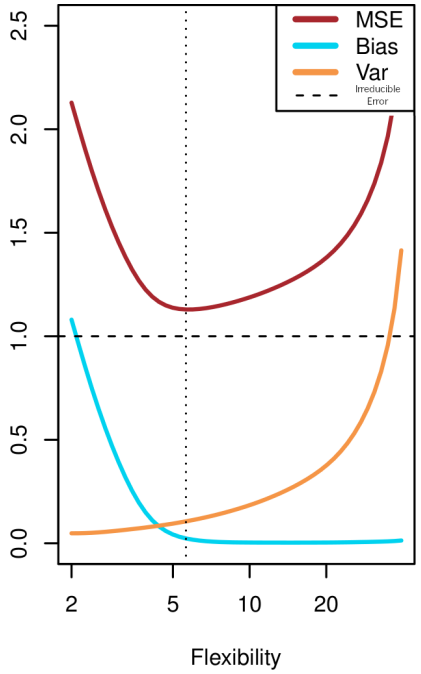


Figure 20.6: Expected test error decomposition, artificial dataset [5].

Let us summarize the main take-aways from the first section:

- the optimal regression function $Y = f(\vec{X}) + \varepsilon$ for **numerical responses** is
$$f(x) = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}];$$
- models are **learned** on training data Tr;

- in practice, we learn the best model from a **restricted** group of model families;
- the best model $\hat{f}(\mathbf{x})$ minimizes the reducible part of the prediction error MSE_{Te}, **evaluated** on testing data Te;
- the **bias-variance trade-off** tells us that models that are too simple (or too rigid) **underfit** the data, and that models that are too complex (or too "loose") **overfit** the data;
- the total prediction error on Te is **bounded below** by the irreducible error.

Finally, remember that a **predictive** model's performance **can only be evaluated on unseen data** (i.e., on data not drawn from the training set Tr); if this requirement is not met, the model is at best **descriptive**.

20.2 Regression Modeling

In the regression setting, the goal is to estimate the regression function

$$f(\mathbf{x}) = \text{E}[Y \mid \vec{X} = \mathbf{x}],$$

the solution to the regression problem

$$Y = f(\vec{X}) + \varepsilon.$$

The best estimate $\hat{f}$ is the model that minimizes

$$\text{MSE}_{\text{Te}}(\hat{f}) = \text{Avg}_{\mathbf{x}^* \in \text{Te}} \text{E}\left[(y^* - \hat{f}(\mathbf{x}^*))^2\right].$$

In practice, this can be hard to achieve without restrictions on the functional form of $\hat{f}$, so we try to **learn** the best $\hat{f}$ from **specific families of models**. Remember, however, that no matter what the approximation function $\hat{f}$ is, we have:[18]

18: Assuming that $\text{Var}(\varepsilon)$ is constant in $\mathbf{x}$.

$$\text{MSE}_{\text{Te}}(\hat{f}) \geq \text{Var}(\varepsilon).$$

What else can we say about $\hat{f}$? In the **ordinary least square framework** (OLS), we assume that

$$\hat{f}_{\text{OLS}}(\mathbf{x}) \approx \mathbf{x}^\top \boldsymbol{\beta},$$

which is to say that we assume that $\hat{f}_{\text{OLS}}$ is nearly globally linear.[19]

19: We neglect the intercept term, in this interpretation.

The true regression function is almost never linear, but the linear assumption yields models $\hat{f}$ that are both **conceptually** and **practically** useful – the model $\hat{f}$ is easily interpretable, and the associated prediction error $\text{MSE}_{\text{Te}}(\hat{f})$ is often "small-ish".

The most common data modeling methods are **linear and logistic regression methods**. By some estimation, 90% of real-world data applications end up using these as their final model, typically after very carefully preparing the data (cleaning, encoding, creation of new variables, transformation of variables, etc.).

That is mostly due to the:

- regression models being straightforward to **interpret** and to **train**;

- MSE_{Te} having a closed-form linear expression, and
- OLS solution being computable using simple matrix manipulations.

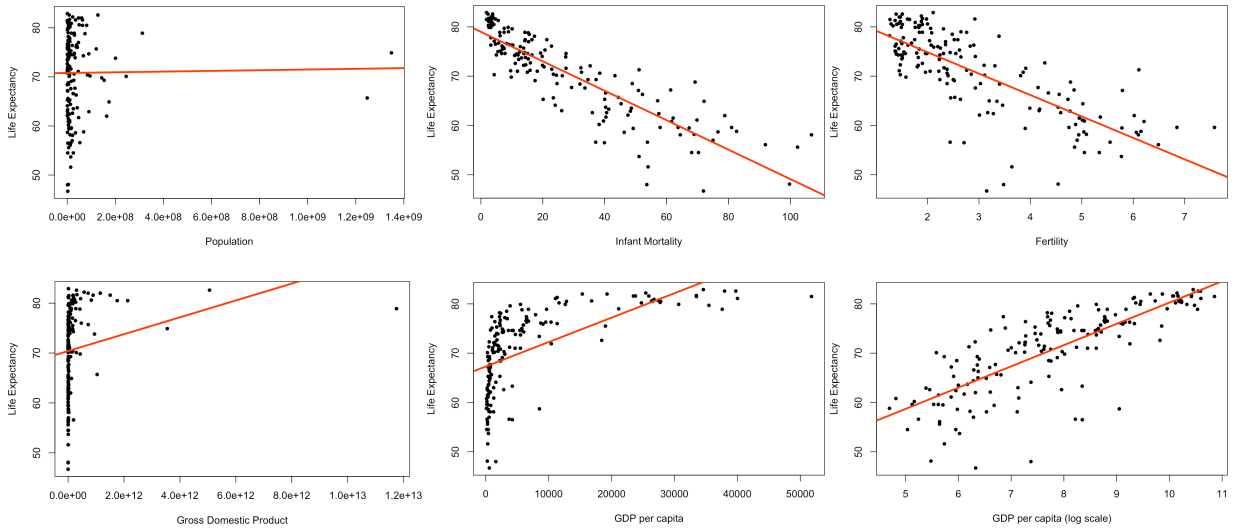
Gapminder Example Let us revisit the `Gapminder` dataset, focusing on observations from 2011.

- Is there a relationship between gross domestic product and life expectancy?
- How strong is the relationship?
- Which factors contribute to the life expectancy?
- How accurately could we predict life expectancy given a set of new observations?
- Is the relationship linear?
- Are there combinations of factors that are linked with life expectancy?

Can the scatterplots of various predictors against life expectancy for the 2011 Gapminder data, shown below with line of best fit, be used to answer these questions?

```
gapminder.2011 <- gapminder.ML |> filter(year==2011)
attach(gapminder.2011)

x=population
y=life_expectancy
plot(x,y, xlab="Population", ylab="Life Expectancy")
abline(lm(y~x), col="red",lwd=3)

x=infant_mortality
y=life_expectancy
plot(x,y, xlab="Infant Mortality", ylab="Life Expectancy")
abline(lm(y~x), col="red",lwd=3)

x=fertility
y=life_expectancy
plot(x,y, xlab="Fertility", ylab="Life Expectancy")
abline(lm(y~x), col="red",lwd=3)

x=gdp
y=life_expectancy
plot(x,y, xlab="Gross Domestic Product",
     ylab="Life Expectancy")
abline(lm(y~x), col="red",lwd=3)

x=gdp/population
y=life_expectancy
plot(x,y, xlab="GDP per capita", ylab="Life Expectancy")
abline(lm(y~x), col="red",lwd=3)

x=log(gdp/population)
y=life_expectancy
plot(x,y, xlab="GDP per capita (log scale)",
```

```
     ylab="Life Expectancy")
abline(lm(y~x), col="red",lwd=3)

detach(gapminder.2011)
```



20.2.1 Formalism

Consider a dataset $\text{Tr} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ with N observations and $p-1$ features. The corresponding **design matrix**, **response vector**, and **coefficient vector** are, respectively,

$$\mathbf{X} = \begin{pmatrix} 1 & x_{1,1} & \cdots & x_{1,p-1} \\ \vdots & \vdots & & \vdots \\ 1 & x_{N,1} & \cdots & x_{N,p-1} \end{pmatrix}, \quad \mathbf{Y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{p-1} \end{pmatrix}.$$

The objective is to find f such that $\mathbf{Y} = f(\mathbf{X}) + \boldsymbol{\varepsilon}$. The OLS solution assumes that $f(\mathbf{X}) = \mathbf{X}\boldsymbol{\beta}$; we must thus learn $\boldsymbol{\beta}$ using the training data Tr.

If $\hat{\boldsymbol{\beta}}$ is an estimate of the true coefficient vector $\boldsymbol{\beta}$, the **linear regression model** associated with Tr is

$$\hat{f}(\mathbf{x}) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_{p-1} x_{p-1}.$$

How do we find $\hat{\boldsymbol{\beta}}$? The OLS estimate minimizes the **loss function**

$$\begin{aligned} \mathscr{L}(\boldsymbol{\beta}) &= \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 = (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})^\top (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) \\ &= \mathbf{Y}^\top \mathbf{Y} - ((\mathbf{X}\boldsymbol{\beta})^\top \mathbf{Y} + \mathbf{Y}^\top \mathbf{X}\boldsymbol{\beta}) + (\mathbf{X}\boldsymbol{\beta})^\top \mathbf{X}\boldsymbol{\beta} \\ &= \mathbf{Y}^\top \mathbf{Y} - (\boldsymbol{\beta}^\top \mathbf{X}^\top \mathbf{Y} + \mathbf{Y}^\top \mathbf{X}\boldsymbol{\beta}) + \boldsymbol{\beta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\beta}. \end{aligned}$$

The loss function is a non-negative symmetric quadratic form in $\boldsymbol{\beta}$, with no restriction on the coefficients, so any minimizer of $\mathscr{L}$ must also be one of its critical points (assuming certain regularity conditions on the data). We are thus looking for coefficients for which $\nabla\mathscr{L}(\boldsymbol{\beta}) = \mathbf{0}$. Since

$$\nabla\mathscr{L}(\boldsymbol{\beta}) = -2(\mathbf{X}^\top \mathbf{Y} - \mathbf{X}^\top \mathbf{X}\boldsymbol{\beta}),$$

any minimizer $\hat{\boldsymbol{\beta}}$ must satisfy the **canonical** (normal) **equations**:

$$\mathbf{X}^\top \mathbf{Y} = \mathbf{X}^\top \mathbf{X}\hat{\boldsymbol{\beta}}.$$

If $\mathbf{X}^\top \mathbf{X}$ is invertible, the minimizer $\hat{\boldsymbol{\beta}}$ is unique and is given by

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{Y}, \quad \text{with} \quad \text{Var}(\hat{\boldsymbol{\beta}}) = \hat{\sigma}^2(\mathbf{X}^\top \mathbf{X})^{-1},$$

where $\hat{\sigma}^2$ is the variance of the residuals.[20] We say that "**we have learned the coefficients $\hat{\boldsymbol{\beta}}$ on the training data Tr using linear regression**".

20: Note that $\mathbf{X}^\top \mathbf{X}$ is a $p \times p$ matrix, which makes the inversion relatively easy to compute even when N is large.

In what follows, we sometimes write $\mathbf{x}$ to represent the observation vector

$$(1, x_1, \ldots, x_{p-1})^\top;$$

it should be clear what is meant from the context.

The **fitted value of the model $\hat{f}$ at input $\mathbf{x}_i \in$ Tr** is

$$\hat{y}_i = \hat{f}(\mathbf{x}_i) = \mathbf{x}_i^\top \hat{\boldsymbol{\beta}},$$

and the **predicted value at an arbitrary $\mathbf{x}^*$** is

$$\hat{y}^* = \hat{f}(\mathbf{x}^*) = \mathbf{x}^{*\top} \hat{\boldsymbol{\beta}}.$$

The **fitted surface** is thus entirely described by the $p + 1$ parameters $\hat{\boldsymbol{\beta}}$; the number of (effective) parameters is a measure of the **complexity** of the learner.

Motivating Example We study a subset of the Gapminder dataset: the observations for 2011, the predictor variables infant mortality X_1 and fertility X_2, and the response variable life expectancy Y. The training data Tr contains $N = 166$ observations and $p = 2$ predictor features.

The design matrix $\mathbf{X}$ is thus of dimension 166×3.

```
library(matlib)
gapminder.2011 <- gapminder.2011 |> dplyr::mutate(const=1)
design.X = gapminder.2011[,c("const","infant_mortality",
                "fertility")]
str(design.X)
```

```
'data.frame':   166 obs. of  3 variables:
 $ const          : num  1 1 1 1 1 1 1 1 1 1 ...
 $ infant_mortality: num  14.3 22.8 106.8 7.2 12.7 ...
 $ fertility      : num  1.75 2.83 6.1 2.12 2.2 1.5 1.88 1.44 1.96 1.9 ...
```

The response is a 166×1 vector.

```
resp.Y = gapminder.2011[,c("life_expectancy")]
```

The constituents of the canonical equations are:

```
(X.t.X = t(as.matrix(design.X)) %*% as.matrix(design.X))
(X.t.Y = t(as.matrix(design.X)) %*% as.matrix(resp.Y))
```

We thus see that

$$\mathbf{X}^\top\mathbf{X} = \begin{pmatrix} 166.0 & 4537.3 & 486.54 \\ 4537.3 & 225043.25 & 18445.28 \\ 486.54 & 18445.28 & 1790.238 \end{pmatrix}$$

and

$$\mathbf{X}^\top\mathbf{Y} = \begin{pmatrix} 11756.7 \\ 291153.33 \\ 32874.95 \end{pmatrix}.$$

We can now compute $\hat{\boldsymbol{\beta}}$:

```
(beta.hat = inv(X.t.X) %*% X.t.Y)
```

Thus,

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{Y} = \begin{pmatrix} 79.677 \\ -0.276 \\ -0.443 \end{pmatrix}.$$

We have seen that the fitted surface is

$$y^* = \hat{f}(\mathbf{x}^*) = 79.677 - 0.276x_1^* - 0.443x_2^*$$

for an observation $\mathbf{x}^* = (x_1^*, x_2^*)$.

Warning: predictions should not be made for observations outside the range (or the envelope) of the training predictors. In this example, the predictor envelope is shown in red in the figure below – one should resist the temptation to predict y^* for $\mathbf{x}^* = (100, 2)$, say.

Least Squares Assumptions Since the family of OLS learners is a subset of all possible learners, the best we can say about $\hat{f}_{\text{OLS}}$ is that

$$\text{MSE}_{\text{Te}}(\hat{f}_{\text{OLS}}) \geq \min_{\hat{f}} \left\{\text{MSE}_{\text{Te}}(\hat{f})\right\} \geq \text{Var}(\varepsilon).$$

In practice, we are free to approximate f with **any** learner $\hat{f}$. If we want $\hat{f}$ to be useful, however, we need to verify that it is a "decent" approximation.

There is another trade-off at play: when we restrict learners to specific families of functions,[21] we typically also introduce a series of assumptions on the data.

21: That is, when we impose structure on the learners.

The OLS assumptions are

- **linearity:** the response variable is a linear combination of the predictors;
- **homoscedasticity:** the error variance is constant for all predictor levels;

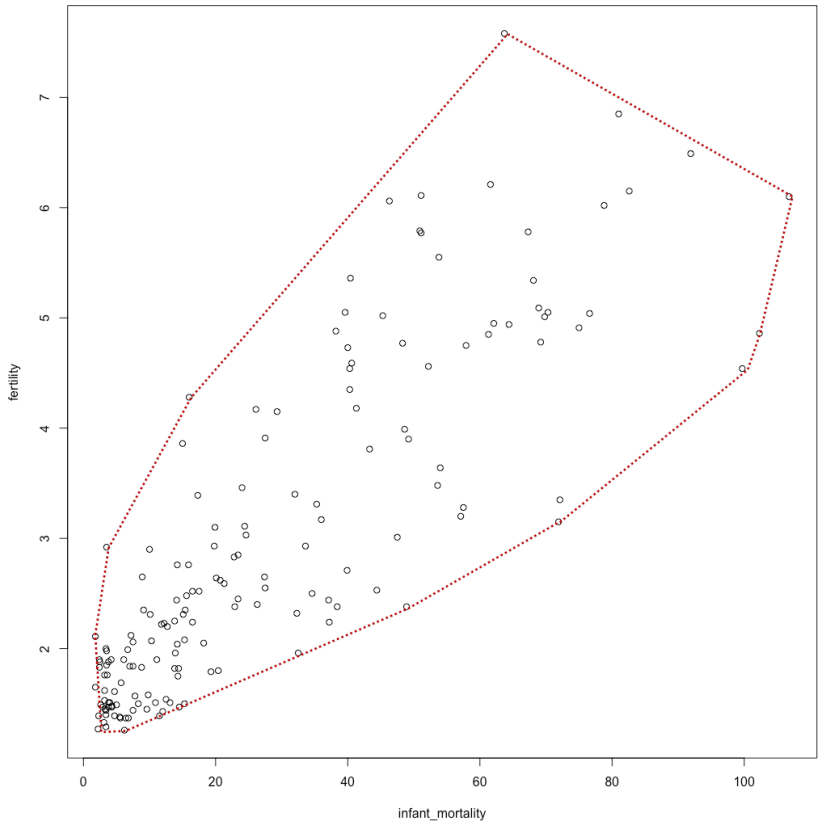


Figure 20.7: Predictor envelope for the Gapminder subset.

- **uncorrelated errors:** the error is uncorrelated from one observation to the next;
- **full column rank for design matrix X:** the predictors are not perfectly multi-collinear;
- **weak exogeneity:** predictor values are free of measurement error.

Mathematically, the assumptions translate to

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

where $\boldsymbol{\beta} \in \mathbb{R}^{p+1}$ is determined on a training set Tr without measurement error, and for which

$$\mathrm{E}[\boldsymbol{\varepsilon} \mid \mathbf{X}] = \mathbf{0} \quad \text{and} \quad \mathrm{E}[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^{\top} \mid \mathbf{X}] = \sigma^2 I_n.$$

Although it is not a requirement, it is also often further assumed that

$$\boldsymbol{\varepsilon} \mid \mathbf{X} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I_n).$$

We will discuss how these assumptions can be generalized at a later stage. In the meantime, however, how can we determine if the choice of model is valid? In the traditional statistical analysis context, there is a number of tests available to the analyst (we will discuss them shortly). In the machine learning context, there is only one real test:

does the model make good predictions?

20.2.2 Least Squares Properties

Let us assume that the OLS assumptions are satisfied. What can we say about the linear regression results? (see Chapter 8 and [6], say, for a refresher).

For the Gapminder example above, for instance, we could us R's `lm()`.

```
f.model = lm(life_expectancy~infant_mortality+fertility)
summary(f.model)
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-15.3233  -2.0057   0.2003   2.9570  10.6370

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)       79.6759     0.7985  99.786   <2e-16 ***
infant_mortality  -0.2763     0.0248 -11.138   <2e-16 ***
fertility         -0.4440     0.4131  -1.075    0.284
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.172 on 163 degrees of freedom
Multiple R-squared:  0.7612,    Adjusted R-squared:  0.7583
F-statistic: 259.8 on 2 and 163 DF,  p-value: < 2.2e-16
```

Coefficient of Determination Let

$$\text{SSE} = \mathbf{Y}^\top[I_n - \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top]\mathbf{Y} = \mathbf{Y}^\top[I_n - \mathbf{H}]\mathbf{Y}$$

and

$$\text{SST} = \mathbf{Y}^\top\mathbf{Y} - n\overline{y}^2.$$

In the Gapminder example, we have:

```
(SSE=anova(f.model)[[2]][3])
```

```
[1] 2837.69
```

```
(SST=as.vector(t(as.matrix(resp.Y)) %*% as.matrix(resp.Y)
     - nrow(as.matrix(resp.Y))*(mean(resp.Y))^2))
```

```
[1] 11882.18
```

The coefficient of determination of the OLS regression is the quotient

$$R^2 = \frac{\text{SST} - \text{SSE}}{\text{SST}} = \frac{\text{Cov}^2(\mathbf{Y}, \mathbf{X}\hat{\boldsymbol{\beta}})}{\sigma_y^2\sigma_{\hat{y}}^2}.$$

In the Gapminder example, we have:

```
(R.2 = 1-SSE/SST)
```

```
[1] 0.761181
```

The coefficient of determination identifies the proportion of the variation of the data explained by the linear regression; as such, $0 \leq R^2 \leq 1$.

If $R^2 \approx 0$, then the predictor variables have little explanatory power on the response; if $R^2 \approx 1$, then the linear fit is deemed to be "good", as a lot of the variability in the response is explained by the predictors. In practice, the number of predictors also affects the goodness-of-fit (this is related to the curse of dimensionality discussed previously).

The quantity

$$R_a^2 = 1 - \frac{N-1}{N-p}(1-R^2) = 1 - \frac{\text{SSE}/(N-p)}{\text{SST}/(N-1)}$$

is the **adjusted coefficient of determination** of the linear regression. While R_a^2 can be negative, it is always smaller than R^2. It also plays a role in the **feature selection** process.

In the Gapminder example, we have:

```
(R.a.2 = 1-(nrow(as.matrix(resp.Y))-1)
     /(nrow(as.matrix(resp.Y))-nrow(X.t.X))*(1-R.2))
```

```
[1] 0.7584
```

This suggests that a fair proportion of the variability in the life expectancy (about 75.7%) is explained by infant mortality and fertility.

Significance of Regression We can determine if at least one of the predictors $X_1, \ldots, X_{p-1}$ is useful in predicting the response Y by pitting

$$H_0 : (\beta_1, \ldots, \beta_{p-1}) = \mathbf{0} \quad \text{against} \quad H_1 : (\beta_1, \ldots, \beta_{p-1}) \neq \mathbf{0}.$$

Under the null hypothesis H_0, the $F-$statistic

$$F^* = \frac{(\text{SST} - \text{SSE})/p}{\text{SSE}/(N-p)} \sim F_{p,N-p}.$$

At significance level α, if $F^* \geq F_{p,N-p;\alpha}$ (the $1-\alpha$ quantile of the F distribution with p and $N-p$ degrees of freedom), then we reject the null hypothesis in favour of the alternative.

In the Gapminder model

$$Y = 79.677 - 0.276X_1 - 0.443X_2 + \varepsilon, \quad N = 166, p = 2,$$

we have:

```
(F.star = ((SST-SSE)/(nrow(X.t.X)))
       /(SSE/(nrow(as.matrix(resp.Y))-nrow(X.t.X))))
```

```
[1] 258.169
```

At a significance level $\alpha = 0.05$, the critical value of the $F_{2,164}$ distribution is:

```
qf(0.05,nrow(X.t.X),nrow(as.matrix(resp.Y))
      -nrow(X.t.X),lower.tail=FALSE)
```

```
[1] 3.051127
```

Since $F^* \geq F_{2,164;0.05}$, at least one of $\beta_1, \beta_2 \neq 0$, with probability 95% (in the frequentist interpretation).

Interpretation of the Coefficients For $j = 1, \ldots, p$, the coefficient β_j is the **average** effect on Y of a 1-unit increase in X_j, **holding all other predictors fixed**. Ideally, the predictors are uncorrelated (such as would be the case in a **balanced design** [10]). Each coefficient can then be tested (and estimated) separately, and the above interpretation is at least reasonable in theory.

In practice, however, we can not always control the predictor variables, and it might be impossible to "hold all other predictors fixed." When the predictors are correlated, there are potential **variance inflation** issues for the estimated regression coefficients, and the interpretation is risky, since when X_j changes, so do the other predictors.[22] More importantly, the interpretation can also be read as a claim of causality, which **should be avoided** when dealing with observational data.

> "The only way to find out what will happen when a complex system is disturbed is to disturb the system, not merely to observe it passively." (paraphrased from [2])

22: If Y represents the total monetary value in a piggy bank, X_1 the number of coins, and X_2 the number of pennies, what is likely to be the sign of β_2 in the model $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon$? Are X_1 and X_2 correlated? What would the interpretation look like, in this case?

In the Gapminder example, the correlation between X_1 and X_2 is:

```
cor(infant_mortality,life_expectancy)
```

```
[1] -0.8714863
```

The predictors are thus strongly correlated, and the standard interpretation is not available to us.

Hypothesis Testing We can also determine if a specific predictor X_j is useful in predicting the response Y, by testing for

$$H_0 : \beta_j = 0 \quad \text{against} \quad H_1 : \beta_j \neq 0.$$

Under the null hypothesis H_0, the test statistic

$$t^* = \frac{\hat{\beta}_j}{\text{se}(\hat{\beta}_j)} \sim T_{N-2},$$

where $\text{se}(\hat{\beta}_j) = \sqrt{\hat{\sigma}^2 (\mathbf{X}^\top \mathbf{X})^{-1}_{j+1,j+1}}$, and $\hat{\sigma}^2 = \frac{\text{SSE}}{N-p}$, and T_{n-2} is the Student T distribution with $N-2$ degrees of freedom.

At a significance level α, if $|t^*| \geq |t_{n-2;\alpha/2}|$ (the $1-\alpha/2$ quantile of the T distribution with $N-2$ degrees of freedom), then we reject the null hypothesis in favour of the alternative.

In the Gapminder model, we have: $N = 166$, $p = 2$, and $\hat{\beta}_1 = -0.276$ so that

```
(sigma.hat.2=SSE/(nrow(as.matrix(resp.Y))
         -nrow(X.t.X)))
```

```
[1] 17.51661
```

```
(se.beta.hat.1=sqrt(sigma.hat.2*inv(X.t.X)[2,2]))
```

```
[1] 0.02488045
```

Thus

```
(t.star=(inv(X.t.X) %*% X.t.Y)[2]/se.beta.hat.1)
```

```
[1] -11.08275
```

At a significance level $\alpha = 0.05$, the critical value of the T_{164} distribution is:

```
qt(0.025,nrow(as.matrix(resp.Y))-2)
```

```
[1] -1.974535
```

Since $|t^*| \geq |t_{164;0.025}|$, $\beta_1 \neq 0$ with probability 95% (in the frequentist interpretation).

Confidence Intervals The **standard error** of $\hat{\beta}_j$ reflects how the estimate would vary under various Tr; it can be used to compute a $(1-\alpha)\%$ **confidence interval** for the true β_j:

$$\text{CI}(\beta_j; 1-\alpha) \equiv \hat{\beta}_j \pm z_{\alpha/2} \cdot \text{se}(\hat{\beta}_j);$$

at $\alpha = 0.05$, $z_{\alpha/2} = 1.96 \approx 2$, so that

$$\text{CI}(\beta_j; 0.95) \equiv \hat{\beta}_j \pm 2\text{se}(\hat{\beta}_j).$$

In the Gapminder example, we have

coeff.	est.	s.e.	t*	95% CI
β_0	79.677	0.7985	99.786	$[78.1, 81.3,]$
β_1	-0.276	0.0248	-11.138	$[-0.33, -0.23]$
β_2	0.443	0.4131	-1.075	$[-1.27, 0.38]$

In **frequentist** statistics, the confidence interval has a particular interpretation – it does not mean, as one might wish, that there is a 95% chance, say, that the true β_j is found in the CI; rather, it suggests that the approach used to build the 95% CI will yield an interval in which the true β_j will reside approximately 95% of the time.[23]

23: Compare with the Bayesian notion of a **credible interval** (see Chapter 25).

The resulting confidence intervals also depend on the underlying model. For instance, the 95% CI for β_1 in the **full model** is $[-0.33, -0.23]$ (see above), whereas the corresponding CI in the **reduced model**

$$\hat{Y} = \gamma_0 + \gamma_1 X_1$$

is $[-0.33, -0.27]$.

The estimates are necessarily distinct as well:

```
reduced.model = lm(life_expectancy ~ infant_mortality)
summary(reduced.model)
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-14.9729  -1.9716   0.1726   2.9727  11.0275

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     78.99279    0.48357  163.35   <2e-16 ***
infant_mortality -0.29888    0.01313  -22.76   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.174 on 164 degrees of freedom
Multiple R-squared:  0.7595,    Adjusted R-squared:  0.758
F-statistic: 517.9 on 1 and 164 DF,  p-value: < 2.2e-16
```

Note that $\hat{\beta}_1 = -0.2763 \neq -0.2989 = \hat{\gamma}_1$.

Feature Selection How would we determine if all the predictors help explain the response Y, or if only a (proper) subset of the predictors is needed? The most direct approach to solve this problem (in the linear regression context) is to run **best subsets** regression.

The procedure is as follows: fit an OLS model for all possible subsets of predictors and select the **optimal model** based on a criterion that balances **training error** with **model size**.

There are 2^{p+1} such models (a quantity that quickly becomes unmanageable). In practice, we need to automate and speed-up the search through a collection of predictor subsets. OLS approaches include **forward selection** and **backward selection** (we discuss these in detail in Chapter 23, *Feature Selection and Dimension Reduction*).

Forward selection is a bottom-up approach:

1. start with the **null model** $\mathcal{M}_0 : Y = \beta_0 + \varepsilon$;
2. fit p simple linear regressions $Y = \beta_0 + \beta_j X_j + \varepsilon$ and add to the null model the predictor X_{j_1} resulting in the lowest SSE:

$$\mathcal{M}_1 : Y = \beta_0 + \beta_{j_1} X_{j_1} + \varepsilon;$$

3. add to that model the predictor X_{j_2} that results in the lowest SSE among all the two-variable models:

$$\mathcal{M}_2 : Y = \beta_0 + \beta_{j_1} X_{j_1} + \beta_{j_2} X_{j_2} + \varepsilon;$$

4. the process continues until a stopping criterion is met.

Backward selection is a top-down approach, and it works in reverse, removing predictors from the full model.

In both approaches, there are at most

$$p + (p-1) + \cdots + 2 + 1 = \frac{p(p+1)}{2} \ll 2^{p+1} \text{ (when } p \text{ is large)}$$

regressions to run. These methods are, frankly, not ideal in the machine learning framework (we will shortly see alternatives).

Other Questions

- How do we handle qualitative variables? (dummy binary variables);
- How do we handle interaction terms? (add features);
- How do we handle outliers? (median regression, Theil-Sen estimate);
- How do we handle non-constant variance of error terms? (data transformations, weighted least square regression, Bayesian regression);
- How do we handle high-leverage observations? (robust regression);
- How do we handle collinearity? (principal component analysis, generalized linear models, partial least square regression);
- How do we handle multiple tests? (Bonferroni correction: for q independent tests with the same data, set significance level to α/q to get joint significance equivalent to α for a single test).

20.2.3 Generalizations of OLS

The OLS assumptions are convenient from a mathematical perspective, but they are not always met in practice.

One way out of this problem is to use **remedial measures** to transform the data into a compliant set; another one is to extend the assumptions and to work out the corresponding mathematical formalism:

- **generalized linear models** (GLM) implement responses with non-normal conditional distributions;
- **classifiers** (logistic regression, decision trees, support vector machines, naïve Bayes, neural networks) extend regression to categorical responses (see Chapter 21);
- **non-linear methods** such as splines, generalized additive models (GAM), nearest neighbour methods, kernel smoothing methods are used for responses that are not linear combinations of the predictors (see Section 20.5);
- **tree-based methods** and **ensemble learning methods** (bagging, random forests, boosting) are used for predictor interactions (see Chapter 21);
- **regularization methods** (ridge regression, LASSO, elastic net) facilitate the process of model selection and feature selection (see the subsection on *Shrinkage Methods*).

Generalized Linear Models GLM extend the OLS paradigm[24] by accommodating response variables with **non-normal** conditional distributions. Apart from the error structure, a GLM is essentially a linear model:

$$Y_i \sim \mathcal{D}(\mu_i), \quad \text{where} \quad g(\mu_i) = \mathbf{x}_i^\top \boldsymbol{\beta}.$$

A GLM consists of:

- a systematic component $\mathbf{x}_i^\top \boldsymbol{\beta}$;
- a random component specified by the distribution $\mathcal{D}$ for Y_i, and
- a link function g.

The **systematic component** is specified in terms of the linear predictor for the i^{th} observation $\eta_i = \mathbf{x}_i^\top \boldsymbol{\beta}$; the general ideas and concepts of OLS carry over to GLM, with the added presence of the link function and the distribution of the response y_i.

In principle, the link function g could be any function linking the linear predictor η_i to the distribution of the response Y_i; in practice, however, g should be **smooth** and **monotonic**.[25]

We could specify any distribution $\mathcal{D}$ for the response Y_i, but they are usually selected from the **exponential family** of distributions.[26] OLS is an example of GLM, with:

- systematic component $\eta_i = \mathbf{x}_i^\top \boldsymbol{\beta}$;
- random component $Y_i \sim \mathcal{N}(\mu_i, \sigma^2)$;
- link function $g(\mu) = \mu$.

24: Ordinary least squares.

25: Or at least differentiable and invertible.

26: These are distributions have probability density functions that satisfy

$$f(\mathbf{x} \mid \vec{\theta}) = h(\mathbf{x}) g(\vec{\theta}) \exp(\vec{\phi}(\vec{\theta}) \cdot \vec{T}(\mathbf{x})).$$

This includes the normal, binomial, Poisson, Gamma distributions, etc. These are all distributions with **conjugate priors** (see Chapter 25).

For a more substantial example, consider the following situation. In the early stages of a rumour spreading, the rate at which new individual learn the information increases exponentially over time. If μ_i is the expected number of people who have heard the rumour on day t_i, a model of the form $\mu_i = \gamma \exp(\delta t_i)$ might be appropriate:

$$\underbrace{\ln(\mu_i)}_{\text{link}} = \ln \gamma + \delta t_i = \beta_0 + \beta_1 t_i = \underbrace{(1, t_i)^\top (\beta_0, \beta_1)}_{\text{systematic component}} .$$

Furthermore, since we measure a count of individuals, the Poisson distribution could be a reasonable choice:

$$Y_i \sim \underbrace{\text{Poisson}(\mu_i),}_{\text{random component}} \qquad \ln(\mu_i) = (1, t_i)^\top (\beta_0, \beta_1).$$

The main advantages of GLM are that:

- there is no need to transform the response Y if it does not follow a normal distribution;
- if the link produces **additive effects**, the assumption of homoscedasticity does not need to be met;
- the choice of the link is separate from the choice of random component, providing modeling flexibility;
- models are still fitted *via* a **maximum likelihood** procedure;
- **inference tools** and **model checks** (Wald ratio test, likelihood ratio test, deviance, residuals, CI, etc.) still apply;
- they are easily implemented (`proc genmod`, `glm()`, etc.), and
- the framework unites various regression modeling approaches (OLS, logistic, Poisson, etc.) under a single umbrella.

20.2.4 Shrinkage Methods

We will discuss the curse of dimensionality (CoD), subset selection, and dimension reduction in Chapter 23. Another approach to dealing with high-dimensionality is provided by the **least absolute shrinkage and selection operator** (LASSO) and its variants.

In what follows, assume that the training set consists of N **centered** and **scaled** observations $\mathbf{x}_i = (x_{i,1}, \cdots, x_{i,p-1})$, with responses y_i.

Let $\hat{\beta}_{\text{OLS},j}$ be the jth OLS coefficient, and set a threshold $\lambda > 0$, whose value depends on the training dataset Tr. Recall that $\hat{\boldsymbol{\beta}}_{\text{OLS}}$ is the exact solution to the OLS problem

$$\hat{\boldsymbol{\beta}}_{\text{OLS}} = \arg\min_{\boldsymbol{\beta}} \{\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2\} = \arg\min_{\boldsymbol{\beta}} \{\text{SSE}\}.$$

In general, **no restrictions** are assumed on the values of the coefficients $\hat{\beta}_{\text{OLS},j}$ – large magnitudes imply that corresponding features **play an important role** in predicting the target. This observation forms the basis of a series of useful OLS variants.

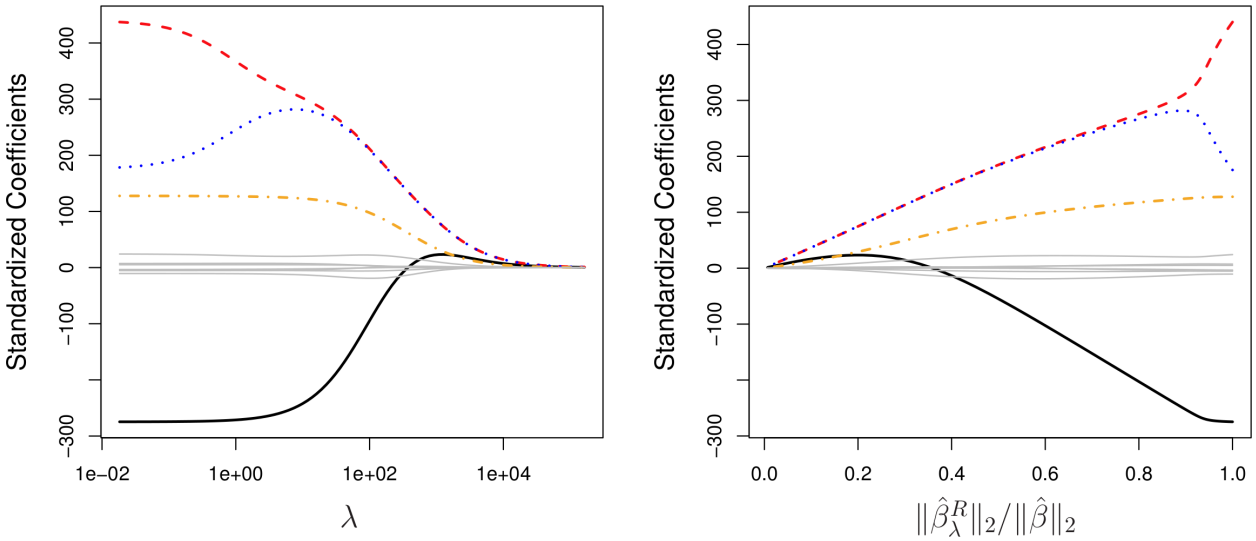


Figure 20.8: Ridge regression coefficients in a generic problem; note how the coefficients converge to 0 when the threshold lambda increases (left); the ratio between the magnitude of the ridge regression parameter and the corresponding OLS parameter is shown on the right [5].

Ridge Regression RR is a method to **regularize** the OLS regression coefficients. Effectively, it shrinks the OLS coefficients by penalizing solutions with large magnitudes – if the magnitude of a specific coefficient is large, then it must have great relevance in predicting the target variable.

This leads to a modified OLS problem:

$$\hat{\boldsymbol{\beta}}_{\text{RR}} = \arg\min_{\boldsymbol{\beta}} \{ \underbrace{\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2}_{\text{SSE}} + \underbrace{N\lambda\|\boldsymbol{\beta}\|_2^2}_{\text{shrinkage penalty}} \}.$$

This quantity is small when SSE is small (i.e., the model is a good fit to the data) and when the **shrinkage penalty** is small (i.e., when each β_j is small). RR solutions are typically obtained *via* numerical methods.[27]

The hyperparameter λ controls the relative impact of both components. If λ is small, then the shrinkage penalty is small even if the individual coefficients β_j are large; if λ is large, then the shrinkage penalty is only small when all coefficients β_j are small (see Figure 20.8).

Setting the "right" value for λ is crucial; it can be done via cross-validation (see [5, pp.227-228] and Section 20.3 (*Cross-Validation*) for details). The OLS estimates are **equivariant**: if $\hat{\beta}_j$ is the estimate for the coefficient β_j of X_j, then $\hat{\beta}_j/c$ is the estimate for the coefficient of the scaled variable cX_j. RR coefficients do not have this property, however, which is why the dataset must be centered and scaled to start with.

Finally, note that RR estimates help to mitigate the bias-variance trade-off and reduce issues related to overfitting.[28]

Regression With Best Subset Selection **BS** runs on the same principle but uses a different penalty term, which effectively sets some of the coefficients to 0 (this could be used to select the features with non-zero coefficients, potentially).

27: For **orthonormal covariates** (which is to say, $\mathbf{X}^\top\mathbf{X} = I_p$), we have, in fact:

$$\hat{\beta}_{\text{RR},j} = \frac{\hat{\beta}_{\text{OLS},j}}{1 + N\lambda}.$$

28: Even if they do not reduce the dimensions of the dataset.

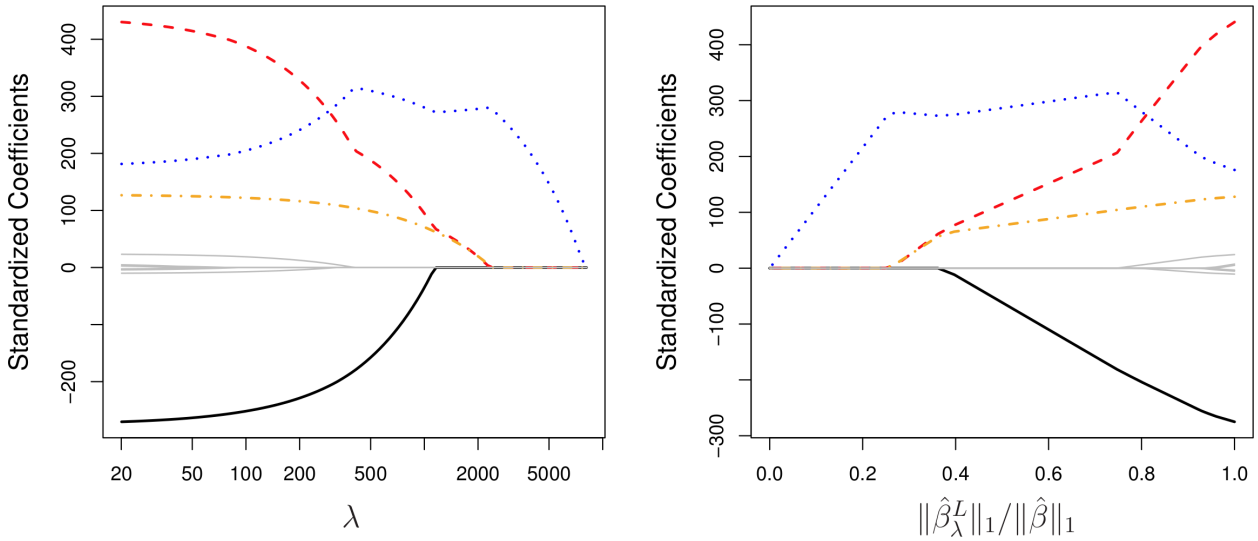


Figure 20.9: LASSO coefficients in a generic problem; note how the coefficients goes directly to 0 after a certain threshold lambda (left); the ratio between the magnitude of the LASSO parameter and the corresponding OLS parameter is shown on the right [5].

The problem consists in solving another modified version of the OLS scenario, namely

$$\hat{\boldsymbol{\beta}}_{\text{BS}} = \arg\min_{\boldsymbol{\beta}}\{\underbrace{\|\mathbf{Y}-\mathbf{X}\boldsymbol{\beta}\|_2^2}_{\text{SSE}} + \underbrace{N\lambda\|\boldsymbol{\beta}\|_0}_{\text{shrinkage}}\}, \qquad \|\boldsymbol{\beta}\|_0 = \sum_j \text{sgn}(|\beta_j|).$$

Solving the BS problem typically (also) requires numerical methods and cross-validation.[29] A slight modification to the RR shrinkage penalty can overcome the lack of equivariance.

29: For orthonormal covariates, we have

$$\hat{\beta}_{\text{BS},j} = \begin{cases} 0 & \text{if } |\hat{\beta}_{\text{LS},j}| < \sqrt{N\lambda} \\ \hat{\beta}_{\text{LS,j}} & \text{if } |\hat{\beta}_{\text{LS},j}| \geq \sqrt{N\lambda} \end{cases}$$

LASSO This approach is an alternative to RR obtained by solving

$$\hat{\boldsymbol{\beta}}_{\text{L}} = \arg\min_{\boldsymbol{\beta}}\{\underbrace{\|\mathbf{Y}-\mathbf{X}\boldsymbol{\beta}\|_2^2}_{\text{SSE}} + \underbrace{N\lambda\|\boldsymbol{\beta}\|_1}_{\text{shrinkage}}\};$$

the penalty effectively forces coefficients which combine the propertiesof RR and BS, selecting at most $\max\{p, N\}$ features, and usually no more than one per group of highly correlated variables (the other coefficients are forced down to 0 when λ is large enough, see Figure 20.9).[30]

30: For orthonormal covariates, we have

$$\hat{\beta}_{\text{L},j} = \hat{\beta}_{\text{OLS},j} \cdot \max\left(0, 1 - \frac{N\lambda}{|\hat{\beta}_{\text{OLS},j}|}\right).$$

Why do we get $\hat{\beta}_{\text{L},j} = 0$ for some j, but not for the RR coefficients? The RR and LASSO formulations are equivalent to

$$\hat{\boldsymbol{\beta}}_{\text{RR}} = \arg\min_{\boldsymbol{\beta}}\{\text{SSE} \mid \|\boldsymbol{\beta}\|_2^2 \leq s\}, \text{ for some } s;$$
$$\hat{\boldsymbol{\beta}}_{\text{L}} = \arg\min_{\boldsymbol{\beta}}\{\text{SSE} \mid \|\boldsymbol{\beta}\|_1 \leq s\}, \text{ for some } s.$$

Graphically, this looks like the images shown in Figure 20.10.

The RR coefficients $\hat{\boldsymbol{\beta}}_{\text{RR}}$ are found at the first intersection of the ellipses of constant SSE around the OLS coefficient $\hat{\boldsymbol{\beta}}$ with the 2−sphere $\|\boldsymbol{\beta}\|_2^2 \leq s$; that intersection is usually away from the axes;[31] this is not usually the case for the intersection of the 1−sphere $\|\boldsymbol{\beta}\|_1 \leq s$.

31: Due to the lack of "sharp" points.

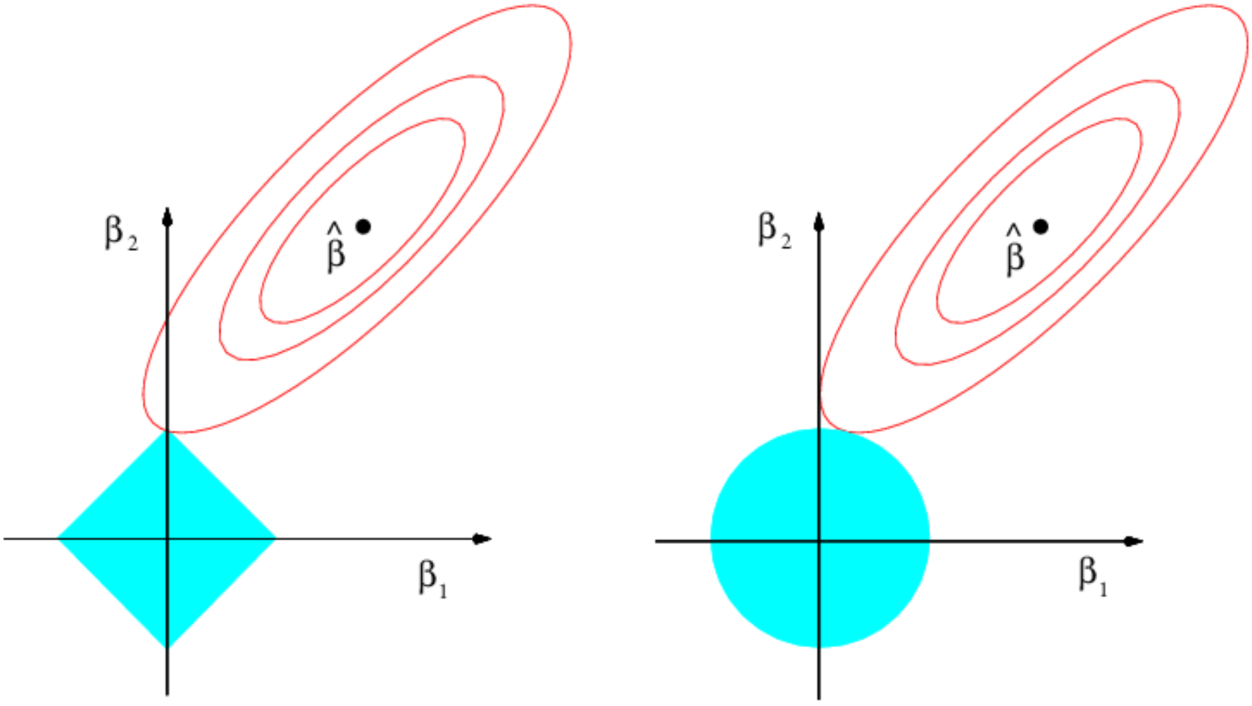


Figure 20.10: Level curves and neighbourhoods for LASSO (left) and ridge regression (right) [5].

The LASSO thus typically produces simpler models, but predictive accuracy matters too (in the form of MSE_{Te}, say).[32]

32: Depending on the data, either of the two approaches can be optimal, thanks to the *No Free Lunch Theorem*.

Generalizations If the response is related to a relatively small number of predictors (whether this is the case or not is not something we usually know *a priori*), LASSO is recommended. The use of other penalty functions (or combinations thereof) provides various extensions, such as: **elastic nets**; **group, fused and adaptive lassos**; **bridge regression**, etc.

The modifications described above were defined assuming an underlying linear regression model, but they generalize to arbitrary regression/classification models as well. For a **loss** (cost) function $\mathscr{L}(\mathbf{Y}, \mathbf{y}(\mathbf{W}))$ between the actual target and the values predicted by the model parameterized by $\mathbf{W}$, and a **penalty** vector $\mathbf{R}(\mathbf{W}) = (R_1(\mathbf{W}), \cdots, R_k(\mathbf{W}))^\top$, the **regularized parametrization** $\mathbf{W}^*$ solves the **general regularization** problem

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \{\mathscr{L}(\mathbf{Y}, \mathbf{y}(\mathbf{W})) + N\boldsymbol{\lambda}^\top \mathbf{R}(\mathbf{W})\},$$

which can be solved numerically, assuming some nice properties on $\mathscr{L}$ and $\mathbf{R}$ [4]; as before, cross-validation can be used to determine the optimal vector $\boldsymbol{\lambda}$ [3].

Gapminder Example In R, regularization is implemented in the package `glmnet` (among others). In `glmnet()` the parameter `alpha` controls the elastic net mixture: LASSO (`alpha = 1`), RR (`alpha = 0`).

Say we are interested in modeling life expectancy Y in the 2011 Gapminder dataset as a function of population, infant mortality, fertility, gdp, and continental membership (we use the entire set as a training set Tr).

A priori, an OLS model on this data would take the form

$$Y = \alpha_0 + \alpha_1 \text{population} + \alpha_2 \text{infant mortality} + \alpha_3 \text{fertility} + \alpha_4 \text{gpd} + \alpha_5 \text{Africa} + \alpha_6 \text{Americas} + \alpha_7 \text{Asia} + \alpha_8 \text{Europe} + \alpha_9 \text{Oceania}.$$

We start by creating dummy variables for the continents:

```
gapminder.2011.f <- fastDummies::dummy_cols(gapminder.2011,
                        select_columns = 'continent')
```

Next, we select the appropriate variables for the response and the training set, and scale and center the data (it must be in a matrix format to be compatible with `glmnet()`:

Setting up the Gapminder dataset

```
library(dplyr)
y <- gapminder.2011.f |> select(life_expectancy) |>
            as.matrix()
x <- gapminder.2011.f |> select(c("population",
            "infant_mortality","fertility","gdp",
            "continent_Africa","continent_Americas",
            "continent_Asia","continent_Europe",
            "continent_Oceania")) |>
            scale(center = TRUE, scale = TRUE) |>
            as.matrix()
```

Finally, we run the regression and extract the LASSO coefficients for hyperparameter $\lambda = 1$:

LASSO coefficients

```
glmnet1 <- glmnet::glmnet(x=x, y=y, type.measure='mse', alpha=1)
(c1 <- coef(glmnet1, x=xx, y=y,s=1,exact=TRUE))
```

```
10 x 1 sparse Matrix of class "dgCMatrix"
                             s1
(Intercept)        70.82349398
population          .
infant_mortality   -5.57897055
fertility           .
gdp                 .
continent_Africa   -1.13074639
continent_Americas  .
continent_Asia      .
continent_Europe    .
continent_Oceania  -0.03096299
```

Thus

$$Y = 70.82 - 5.58(\text{infant mortality}) - 1.13(\text{Africa}) - 0.03(\text{Oceania}).$$

For RR ($\alpha = 0$), we obtain, with the same hyperparameter $\lambda = 1$:

Ridge regression

```
glmnet0 <- glmnet::glmnet(x=x, y=y, type.measure='mse', alpha=0)
(c0 <- coef(glmnet0, x=xx, y=y,s=1,exact=TRUE))
```

```
10 x 1 sparse Matrix of class "dgCMatrix"
                            s1
(Intercept)         70.8234940
population          -0.3471671
infant_mortality    -4.4002779
fertility           -0.6348077
gdp                  0.5803223
continent_Africa    -1.6275714
continent_Americas   0.5475769
continent_Asia       0.6117358
continent_Europe     1.0141934
continent_Oceania   -0.6855980
```

which is to say:

$$Y = 70.82 - 0.34(\text{population}) - 4.4(\text{infant mortality}) - 0.63(\text{fertility}) + 0.58(\text{gdp}) - 1.62(\text{Africa}) + 0.55(\text{Americas}) + 0.61(\text{Asia}) + 1.01(\text{Europe}) - 0.68(\text{Oceania}),$$

which is compatible with the above discussion.

The coefficient values themselves are not as important as their signs and the fact that they are roughly similar in both models.

It is important to note, however, that the choice of $\lambda = 1$ was arbitrary, and that we have not been evaluating the result on test data Te. We will revisit these issues in Section 20.3 (*Cross-Validation*).

20.3 Resampling Methods

How do we determine the variability of a regression fit? It can be done by drawing different samples from the available data, fitting a regression model to each sample, and then examining the extent to which the various fits differ from one another.

Resampling methods provide additional information about a fitted model, by applying the same fitting approach to various sub-samples of the training set Tr. We will consider three such methods:

- **cross-validation**, which estimates the test error associated with a modeling approach in order to evaluate model performance;
- the **bootstrap**, which provides a measure of accuracy, standard deviation, bias, etc. of various model parameter estimates, and
- the **jackknife**, which is a simpler approach with the same aims as the bootstrap.

The **test error** associated with a statistical learning model is the average error arising when predicting the response for observations that were not used to train the model.

The **training error**, on the other hand, is computed directly by comparing the model's predictions to the actual responses in Tr. In general, the training error underestimates the test error, dramatically so when the model complexity increases (see **variance-bias trade-off**, Figure 20.5).

A possible way out of this conundrum is to set aside a large-enough testing set Te, but that's not always possible if the original dataset is not that large in the first place.[33]

33: Some methods make direct adjustments to the training error rate in order to estimate the test error (e.g., Mallow's C_p statistic, R^2_a, AIC, BIC, etc.)

In the statistical learning framework, we estimate the test error by **holding a subset** Va $\subseteq$ Tr **out** from the fitting process (which takes place on Tr $\setminus$ Va). The **validation approach** is a simple strategy that is used to estimate the test error associated with a particular statistical model on a set of observations.

Formally, the latter is split into a **training set** Tr and a **validation set** Va (the hold-out set). The model is fit on the training set; the fitted model is used to make predictions on the validation set. The resulting validation set error provides an estimate for the test error.

This approach is easy to implement and interpret, but it has a number of drawbacks, most importantly:

- the validation error is highly dependent on the choice of the validation set, and is thus quite **volatile**;
- the model is fitted on a proper subset of the available observations, and we might expect that this would lead to **the validation error being larger than the test error in general**, and
- a number of classical statistical models can provide test error estimates without having to resort to the validation set approach.

20.3.1 Cross-Validation

K**-fold cross-validation** is a widely-used approach to estimate the test error without losing some observations to a hold-out set.[34]

34: It can also provide a basis for model selection.

The procedure is simple:

1. Divide the dataset **randomly** into K (roughly) equal-sized **folds** (typically, $K = 4, 5, 10$).
2. Each fold plays, in succession, the role of the **validation set**. If there are N observations in the dataset, partition
$$\{1, \dots, N\} = \underbrace{\mathscr{C}_1}_{\text{fold 1}} \sqcup \cdots \sqcup \underbrace{\mathscr{C}_K}_{\text{fold } K}.$$
If $|\mathscr{C}_k| = n_k$, we expect $n_k \approx \frac{N}{K}$ for all $k = 1, \dots, K$.
3. For all $k = 1, \dots, K$, fit a model on observations $\{1, \dots, N\} \setminus \mathscr{C}_k$ and denote the error on $\mathscr{C}_k$ by E_k.[35]
4. Write $\overline{E}$ for the average of the E_k.

35: For a regression model, there are many options but we typically use
$$E_k = \sum_{i \in \mathscr{C}_k} \frac{(y_i - \hat{y}_i)^2}{n_k}.$$

5. The **cross-validation estimate** of the test error is

$$\mathrm{CV}_{(K)} = \sum_{k=1}^{K} \frac{n_k}{N} E_k,$$

with standard error

$$\widehat{\mathrm{se}}\left(\mathrm{CV}_{(K)}\right) = \sqrt{\frac{1}{K-1}\sum_{k=1}^{K}(E_k - \overline{E})^2}.$$

These steps could also be **replicated** n times to generate a distribution of an evaluation metric, such as the standard error.[36]

36: See Figure 19.33 for an illustration.

The resulting mean can prove useful in order to determine how well a statistical learning procedure will perform on unseen data. If, however, we are interested in selecting a method from a list of methods, or a flexibility level among a family of approaches, we do not care about the specific value of $\mathrm{CV}_{(K)}$ so much as where it is minimized.[37]

37: The estimate is usually biased, anyway.

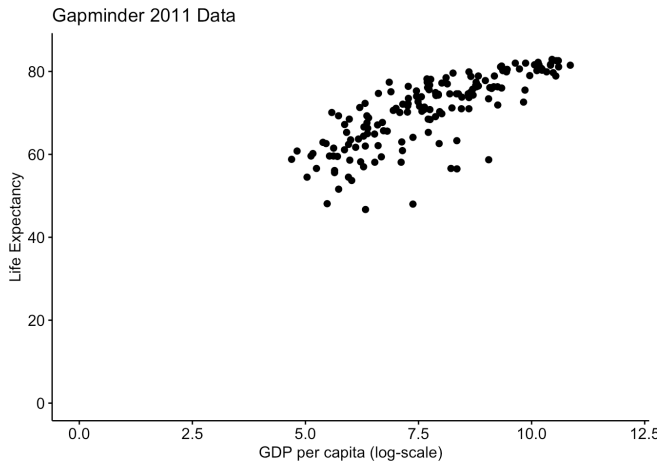
From the perspective of **bias reduction** (in the estimate for the test error), the best choice is $K = N$, but this is mitigated by the variance-bias trade-off. With $K = N$, we have N models and N estimates for the test error, but these estimates are **highly correlated** and the mean of highly correlated estimates has **high variance**.[38]

38: See Section 20.3, *Jackknife*, for details.

Gapminder Example We use cross-validation in the `Gapminder` dataset to estimate the test error $\mathrm{MSE}_{\mathrm{Te}}$ when predicting life expectancy as a regression against the logarithm of the GDP per capita for the 2011 data.

Gapminder subset

```
gapminder.2011.cv <- gapminder.2011 |>
  dplyr::mutate(lgdppc = log(gdp/population)) |>
  select(life_expectancy,lgdppc)

ggpubr::ggscatter(gapminder.2011.cv, x="lgdppc",
        y="life_expectancy", palette="jco", size = 2,
        xlab="GDP per capita (log-scale)", xlim=c(0,12),
        ylab = "Life Expectancy",  ylim=c(0,85),
        title = "Gapminder 2011 Data")
```



We split the dataset into $K = 10$ random folds, each containing 16 or 17 observations, and fit 10 linear regression models using the 149 or 150 remaining observations.[39]

39: Note that the estimates for β_0, β_1, and MSE_{Te} are likely to be correlated from one fold to the next, since the respective training sets share a fair number of observations.

The indices for each of the folds are computed below:

Setting-up the folds

```
set.seed(0) # for replicability
true.order = sample.int(nrow(gapminder.2011.cv),
        nrow(gapminder.2011.cv),replace=FALSE)

index=list()
for(k in 1:6){
  index[[k]] = true.order[((k-1)*17+1):(k*17)]
}
for(k in 7:10){
  index[[k]] = true.order[(102+(k-6-1)*16+1):(k*16+6)]
}
```

Each fold is used, in turn, as a testing set while the remaining folds form the training set. We fit an OLS model on each training set, and evaluate the MSE performance of the model on the appropriate fold testing set.

Compute the test MSE for each fold

```
training.gap = list()
testing.gap = list()
model.lm.gap = list()
pred.lm.gap = list()
beta.0 = c()
beta.1 = c()
MSE.cv = c()
n.row = c()

for(k in 1:10){
  n.row[k]=length(index[[k]])
  training.gap[[k]] = gapminder.2011.cv[-index[[k]],]
  testing.gap[[k]] = gapminder.2011.cv[index[[k]],]
  model.lm.gap[[k]] = lm(life_expectancy~lgdppc,
            data=training.gap[[k]])
  beta.0[k] = model.lm.gap[[k]][[1]][1]
  beta.1[k] = model.lm.gap[[k]][[1]][2]
  pred.lm.gap[[k]] = predict(model.lm.gap[[k]],
            newdata=testing.gap[[k]])
  tmp = data.frame(pred.lm.gap[[k]],testing.gap[[k]][1])
  MSE.cv[k] = 1/nrow(tmp)*sum((tmp[,1]-tmp[,2])^2)
}
```

The number of observations in each fold, as well as the regression parameters and the MSE on each fold testing set are shown below:

```
(results = data.frame(n.row,beta.0,beta.1,MSE.cv))
```

n.row	beta.0	beta.1	MSE.cv
17	37.69812	4.254105	33.859051
17	36.22257	4.442061	21.415376
17	37.59386	4.247255	45.620933
17	36.66761	4.345484	29.584469
17	37.49685	4.268917	24.127300
17	36.49849	4.386124	19.398769
16	36.78991	4.380887	48.157391
16	36.91113	4.331365	23.142625
16	37.41767	4.274771	7.837172
16	37.68955	4.254600	19.743046

The 10−fold cross-validation estimate of MSE_{Te} is thus

$$\overline{\text{MSE}_{\text{Te}}} = \frac{1}{10}\sum_{k=1}^{10}\text{MSE}_{\text{Te}_k} = 27.29;$$

$$\text{CV}_{(K)} = \sum_{k=1}^{10}\frac{n_k}{166}\text{MSE}_{\text{Te}_k} = 27.35;$$

$$\widehat{\text{se}}\left(\text{CV}_{(K)}\right) = \sqrt{\frac{1}{10-1}\sum_{k=1}^{10}(\text{MSE}_{\text{Te}_k} - \overline{\text{MSE}_{\text{Te}}})^2} = 12.38;$$

these can be computed as below.

CV results

```
mean.MSE = mean(results$MSE.cv)
cv.k = sum(results$n.row*results$MSE.cv/sum(results$n.row))
se.cv.k = sqrt(1/(nrow(results)-1)*sum((results$MSE.cv-
                              mean.MSE)^2))
```

Thus, $27.35 \pm 2(12.38) \equiv (2.59, 52.11)$ is a 95% CI for the MSE_{Te}.

We can also get 10−fold cross-validation estimates of β_0, β_1: we have

$$\beta_{0_{(K)}} = \sum_{k=1}^{10}\frac{n_k}{166}\beta_{0;k} = 37.10$$

$$\widehat{\text{se}}\left(\beta_{0_{(K)}}\right) = \sqrt{\frac{1}{10-1}\sum_{k=1}^{10}(\beta_{0;k} - \overline{\beta_0})^2} = 0.54,$$

so $\text{CI}(\beta_0; 0.95) \equiv 37.10 \pm 2(0.54) \equiv (36.00, 38.18)$ and

$$\beta_{1_{(K)}} = \sum_{k=1}^{10}\frac{n_k}{166}\beta_{1;k} = 4.32$$

$$\widehat{\text{se}}\left(\beta_{1_{(K)}}\right) = \sqrt{\frac{1}{10-1}\sum_{k=1}^{10}(\beta_{1;k} - \overline{\beta_1})^2} = 0.07,$$

so $\text{CI}(\beta_1; 0.95) \equiv 4.32 \pm 2(0.07) \equiv (4.18, 4.56)$, as computed below.

CV estimates for the regression coefficients

```
mean.beta_0 = mean(results$beta_0)
cv.beta_0.k = sum(results$n.row*results$beta_0/
            sum(results$n.row))
se.cv.beta_0.k = sqrt(1/(nrow(results)-1)*sum(
                (results$beta_0-mean.beta_0)^2))

mean.beta_1 = mean(results$beta_1)
cv.beta_1.k = sum(results$n.row*results$beta_1/
            sum(results$n.row))
se.cv.beta_1.k = sqrt(1/(nrow(results)-1)*sum(
                (results$beta_1-mean.beta_1)^2))
```

LASSO and Regression Ridge Revisited How would we pick the optimal hyperparameter λ in shrinkage regressions? Let us revisit the example from Section 20.2 (*Shrinkage Methods*).

As before, we are interested in modeling life expectancy Y in the 2011 Gapminder dataset as a function of population, infant mortality, fertility, gdp, and continental membership.[40] We run a 5-fold cross-validation LASSO regression for a variety of hyperparameter values λ, and evaluate the CV test error for each λ using MSE. The optimal λ is the one that minimizes the CV test error.

40: We use `gapminder.2011.f`, `x`, and `y` as in that section.

Let us start with the LASSO (`alpha=1`):

```
glmnet1 <- glmnet::cv.glmnet(x=x, y=y, type.measure='mse',
            nfolds=5, alpha=1)
(c1 <- coef(glmnet1, s='lambda.min', exact=TRUE))
```

```
                          s1
(Intercept)        70.8234940
population          .
infant_mortality   -5.7375945
fertility           .
gdp                 0.1616446
continent_Africa   -1.7592037
continent_Americas  .
continent_Asia      .
continent_Europe    0.1219114
continent_Oceania  -0.7977736
```

The optimal λ in this case is:

```
(lambda1 = glmnet1$lambda.min)
```

```
[1] 0.3118295
```

We repeat the process for RR (`alpha = 0`):

```
glmnet0 <- glmnet::cv.glmnet(x=x, y=y, type.measure='mse',
            nfolds=5, alpha=0)
(c0 <- coef(glmnet0, s='lambda.min', exact=TRUE))
```

```
                            s1
(Intercept)         70.8234940
population          -0.3466483
infant_mortality    -4.6968992
fertility           -0.4240814
gdp                  0.5813385
continent_Africa    -1.6192452
continent_Americas   0.5467797
continent_Asia       0.6295896
continent_Europe     1.0091460
continent_Oceania   -0.7207190
```

The optimal λ in this case is:

```
(lambda0 = glmnet0$lambda.min)
```

```
[1] 0.7373175
```

Cross-Validation with `Python` Let us take a look at how we could estimate the test error *via* cross-validation manually in `Python`. The following modules will be necessary: `statsmodels` to run linear models (in particular to define **formulas** for linear regression), `numpy` for numerical operations, and `pandas` for data frame manipulations.

Python modules for CV

```
import statsmodels.formula.api as smf
import numpy as np
import pandas as pd
import random
random.seed(0) # for replicability
```

We use the `calculus.csv` dataset from Section 1.6, whose structure is as shown below. We will try to predict students' grades in terms of the other predictors, using linear regression. In particular, we are interested in which model does a better job of predicting the grades.

```
df = pd.read_csv('calculus.csv')
df.head()
```

```
      ID Sex  Grade   GPA  Year
0  10001   F     47  5.02     2
1  10002   M     57  3.82     1
2  10003   M     91  7.70     1
3  10004   M     71  4.82     1
4  10005   F     83  7.91     1
```

We start by obtaining a random permutation of the observations (the `pandas` method `iloc()` selects values for specified indices).

```
nrows = len(df)
permuted = df.iloc[np.random.permutation(nrows)]
permuted.head()
```

```
      ID Sex  Grade    GPA  Year
60  10061   F     97  11.45     2
61  10062   M     70   3.65     1
28  10029   M     98  11.90     1
49  10050   F     92  11.05     1
50  10051   M     79   6.87     2
```

In this example, we separate the sample indices into $k = 5$ folds for cross-validation using the `numpy` function `array_split()`.

```
k = 5
chunks = np.array_split(range(nrows), k)
```

We iterate over each fold as a **test set** while using the remaining folds as a **training set**.

Say `chunk[i]` is the current test set; we can obtain the corresponding training set as follows:

```
training = permuted.iloc[ np.concatenate( [ chunks[j]
               for j in range(k) if j != i]) ]
```

We then perform a linear regression over this training set (with the `statsmodels` methods `ols()` and `fit()`) and compute the MSE over the test set using the predicted values. Remember, this is for a single fold:

```
fit = smf.ols(formula=m, data = training).fit()
test = permuted.iloc[chunks[i]]
pred = fit.predict( test )
testerror = ((pred - test['Grade'])**2).mean()
```

In the chunk of code above, `formula=m` is an R-style formula. In the following, we go through a number of possible formulas, for all folds.

```
f = ['Grade ~ GPA + C(Year) + C(Sex)',
     'Grade ~ GPA + C(Year)',
     'Grade ~ GPA + C(Sex)', 'Grade ~ GPA' ]

for m in f:
    testerror = 0.0
    for i in range(k):
        training = permuted.iloc[ np.concatenate(
            [ chunks[j] for j in range(k) if j != i]) ]
```

```
        fit = smf.ols(formula=m, data = training).fit()
        test = permuted.iloc[chunks[i]]
        pred = fit.predict( test )
        testerror += ((pred - test['Grade'])**2).mean()
    testerror /= k
    print(testerror, m)
```

```
118.2165188650409 Grade ~ GPA + C(Year) + C(Sex)
117.4815061224269 Grade ~ GPA + C(Year)
115.77980266850878 Grade ~ GPA + C(Sex)
114.87270405373037 Grade ~ GPA
```

The best model is given by the formula `Grade ~ GPA`.

20.3.2 Bootstrap

The **bootstrap procedure** uses re-sampling of the available data to **mimic the process of obtaining new replicates**, which allows us to estimate the variability of a statistical model parameter of interest **without the need to generate new observations**.

Replicates are obtained by repeatedly sampling observations from the original dataset **with replacement**. A **bootstrap dataset** Tr^* for a training set Tr with N observations is a sample of N such observations, drawn **with replacement**.

The process is repeated M times to obtain bootstrap samples Tr_i^* and parameter estimates $\hat{\alpha}_i^*$, for $i = 1, \ldots, M$, from which we derive a **bootstrap estimate**

$$\hat{\alpha}^* = \frac{1}{M} \sum_{i=1}^{M} \hat{\alpha}_i^*,$$

with standard error

$$\widehat{\text{se}}\,(\hat{\alpha}^*) = \sqrt{\frac{1}{M-1} \sum_{i=1}^{M} (\hat{\alpha}_i^* - \hat{\alpha}^*)^2}.$$

The bootstrap can also be used to build **approximate frequentist confidence intervals** for the parameter α.[41] We can even construct a covariance structure for the parameters, given enough replicates.

Finally, it should be noted that in more complex scenarios, the appropriate bootstrap procedure might be more sophisticated than what has been described here.[42]

41: Note that this is not as straightforward as one might think, so caution is advised.

42: For instance, sampling with replacement at the observation level would not preserve the covariance structure of time series data.

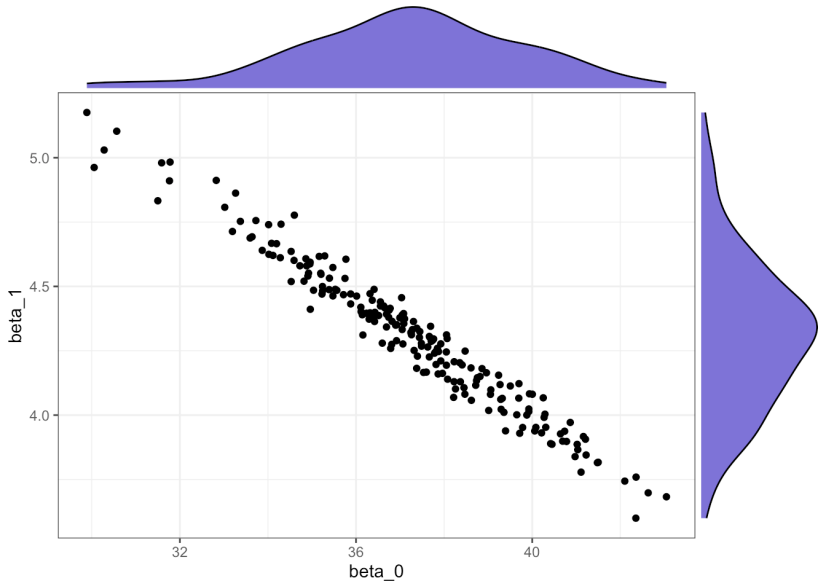
Gapminder Example We use the bootstrap procedure for the regression problem with life expectancy and the log of the GDP per capita in the 2011 Gapminder data.

We draw, with replacement, $M = 200$ bootstrap samples of size $N = 166$ from the original dataset. For each sample $1 \leq i \leq M$, we find the OLS fit and retain the intercept $\beta_{0,i}$ and slope $\beta_{1,i}$.

```
beta_0 = c()
beta_1 = c()

set.seed(0) # for replicability
for(k in 1:200){
  index = sample.int(nrow(gapminder.2011.cv),
        nrow(gapminder.2011.cv),replace=TRUE)
  training.gap = gapminder.2011.cv[-index,]
  model.lm.gap = lm(life_expectancy~lgdppc,
                    data=training.gap)
  beta_0[k] = model.lm.gap[[1]][1]
  beta_1[k] = model.lm.gap[[1]][2]
}

results.boot = data.frame(beta_0,beta_1)
```

We display the joint distribution of $\boldsymbol{\beta} = (\beta_0, \beta_1)^\top$, together with the marginal distributions for each parameter.

```
library(ggplot2)
p <- ggplot(results.boot, aes(x=beta_0, y=beta_1)) +
  geom_point() +
  theme(legend.position="none")
ggExtra::ggMarginal(p, type="density", fill = "slateblue")
```



We see that $\boldsymbol{\beta}$ roughly follows a multivariate normal $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with

$$\boldsymbol{\mu} \approx \hat{\boldsymbol{\mu}}^* = \begin{pmatrix} 37.22 \\ 4.31 \end{pmatrix}, \; \boldsymbol{\Sigma} \approx \hat{\boldsymbol{\Sigma}}^* = \begin{pmatrix} 6.32 & -0.72 \\ -0.72 & 0.08 \end{pmatrix},$$

as computed below:

```
boot.beta_0 = mean(results.boot$beta_0)
boot.beta_1 = mean(results.boot$beta_1)
cov(results.boot)
```

The vector $\hat{\boldsymbol{\mu}}^*$ provides the bootstrap estimates; the corresponding estimates for the standard errors are $\widehat{\text{se}}(\hat{\boldsymbol{\mu}}^*) = (2.51, 0.29)^\top$, and

$$\begin{aligned}\text{CI}(\beta_0; 0.95) &= 37.22 \pm 2(2.51) \equiv (32.19, 42.25),\\ \text{CI}(\beta_1; 0.95) &= 4.31 \pm 2(0.29) \equiv (3.73, 4.89);\end{aligned}$$

the standard errors are computed as below:[43]

43: Note that the bootstrap CI are wider than the corresponding cross-validation CI.

```
se.boot.beta_0 = sqrt(1/(nrow(results.boot)-1)*
  sum((results.boot$beta_0-mean(results.boot$beta_0))^2))
se.boot.beta_1 = sqrt(1/(nrow(results.boot)-1)*
  sum((results.boot$beta_1-mean(results.boot$beta_1))^2))
```

20.3.3 Jackknife

The **jackknife estimator** arises from cross-validation when $K = N$;[44] the sole difference being in the standard error estimate

$$\widehat{\text{se}}(\hat{\alpha}^*) = \sqrt{\frac{N-1}{N}\sum_{i=1}^{N}(\hat{\alpha}_i^* - \overline{\hat{\alpha}^*})^2}.$$

44: The jackknife procedure is also known as **leave one out validation**.

Gapminder Example We use the jackknife procedure on the same task as in the previous section.

For each fold $1 \leq k \leq N$, we find the OLS fit on TR_k and retain the intercept $\beta_{0,k}$ and slope $\beta_{1,k}$. The code is exactly as in the bootstrap case, with one exception: we replace the line

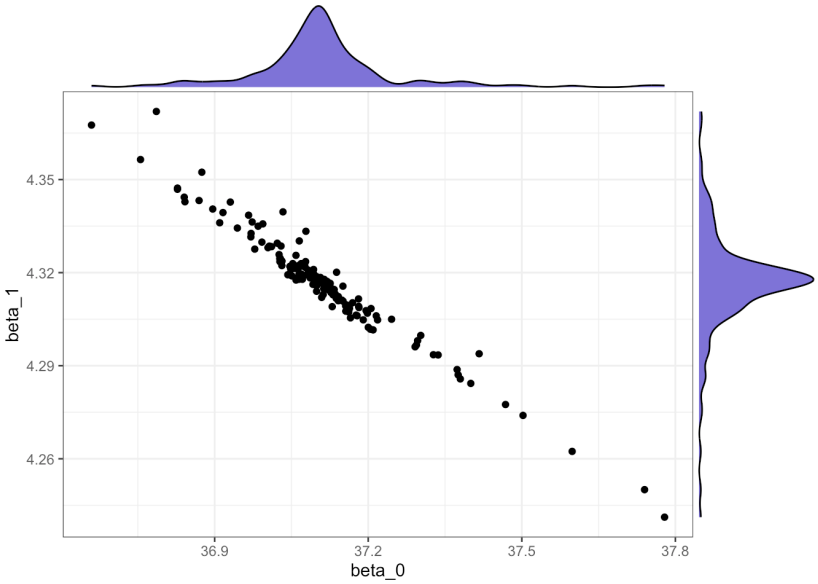
```
  index = sample.int(nrow(gapminder.2011.cv),
        nrow(gapminder.2011.cv),replace=TRUE)
```

by

```
  index = k
```

We display the joint distribution of $\boldsymbol{\beta} = (\beta_0, \beta_1)^\top$, together with the marginal distributions for each parameter.

```
library(ggplot2)
p <- ggplot(results.jack, aes(x=beta_0, y=beta_1)) +
  geom_point() +
  theme(legend.position="none")
ggExtra::ggMarginal(p, type="density", fill = "slateblue")
```



We see that $\boldsymbol{\beta}$ roughly a multivariate normal $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with

$$\boldsymbol{\mu} \approx \hat{\boldsymbol{\mu}}^* = \begin{pmatrix} 37.11 \\ 4.32 \end{pmatrix}, \; \boldsymbol{\Sigma} \approx \hat{\boldsymbol{\Sigma}}^* = \begin{pmatrix} 0.021 & -0.002 \\ -0.002 & 0.0003 \end{pmatrix},$$

as can be computed below:

```
jack.beta_0 = mean(results.jack$beta_0)
jack.beta_1 = mean(results.jack$beta_1)
cov(results.jack)
```

The vector $\hat{\boldsymbol{\mu}}^*$ provides the jackknife estimates; the corresponding estimates for the standard errors are $\widehat{\text{se}}(\hat{\boldsymbol{\mu}}^*) = (1.86, 0.21)^\top$, and

$$\begin{aligned} \text{CI}(\beta_0; 0.95) &= 37.11 \pm 2(1.86) \equiv (33.38, 40.83); \\ \text{CI}(\beta_1; 0.95) &= 4.32 \pm 2(0.21) \equiv (3.890, 4.744). \end{aligned}$$

The standard errors are computed as below:

```
se.jack.beta_0 = sqrt(1/nrow(results.jack)*
  (nrow(results.jack)-1)*sum((results.jack$beta_0-
     mean(results.jack$beta_0))^2))
se.jack.beta_1 = sqrt(1/nrow(results.jack)*
  (nrow(results.jack)-1)*sum((results.jack$beta_1-
    mean(results.jack$beta_1))^2))
```

In this case, the jackknife estimates are tighter than the corresponding bootstrap estimates, but looser than the cross-validation estimates. Will this always be the case?

20.4 Model Selection

A linear model

$$Y = \vec{X}^\top \boldsymbol{\beta} + \varepsilon$$

should be seen as an attempt to approximate the regression function

$$y = f(\mathbf{x}) = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}].$$

But what we gain in convenience of fit (and structure) by using a linear model, we may lose in modeling accuracy.

In this context, we assume a **linear relationship** between the **response** Y and the **predictors** $X_1, \ldots, X_p$, which we (typically) fit using the **(ordinary) least squares** (OLS) framework, which is to say

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \{\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2\},$$

for the **response vector Y** and **design matrix X** provided by a **training set** Tr; additional assumptions on the **error components** $\boldsymbol{\varepsilon}$ usually require

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma \mathbf{I}_N),$$

where n represents the number of observations in Tr.

Fundamentally, there are 3 ways in which the OLS framework can be extended:

1. additive but non-linear models (see Section 20.5, *Generalized Additive Models*);
2. non-linear models (see Section 20.5 and Chapter 21), and
3. replacing LS with alternative fitting procedures (see Section 20.2, *Shrinkage Methods*).

The latter approach can produce **better accuracy** than OLS without sacrificing too much in the way of **model interpretability**.[45]

45: In practice, linear models have distinct advantages over more sophisticated models, mainly in the areas of superior interpretability and (frequently) appropriate predictive performances (especially for linearly separable data). These "Old Faithful" models will still be there if fancy deep learning models fail analysts in the future.

But in the OLS framework, prediction accuracy suffers when $p > n$, due to **curse of dimensionality** (see Section 23.2.2, *Curse of Dimensionality*); model interpretability can be improved by removing **irrelevant features** or by reducing p.

The 3 classes of methods to do so are:

- shrinkage and regularization methods;
- dimension reduction, and
- subset selection/feature selection.

For **shrinkage/regularization methods**, we fit a model involving all p predictors, but the estimated coefficients are shrunk towards 0 relative to the OLS parameter estimates, which has the effect of reducing variance and simultaneously perform variable selection (see Section 20.2, *Shrinkage Methods*).

In **dimension reduction**, we project the p predictors onto a manifold $\mathcal{H}$, with $\dim(\mathcal{H}) = m \ll p$; in numerous circumstances, $\mathcal{H}$ is a subspace of $\mathbf{R}^p$ and we can fit an OLS model on the projected coordinates (see Section 23.2, *Dimension Reduction*).

In **subset selection**, we identify a subset of the p predictors for which there is evidence of a (strong-ish) link with the response, and we fit a model to this reduced set using the OLS framework. Given p predictors (some of which may be interaction terms), there are 2^p OLS models that can be fit on a training set Tr.

Which of those models should be selected as the **best model**?

20.4.1 Best Subset Selection

In the **best subset selection** BSS approach, the search for the best model is usually broken down into 3 stages:

1. let $\mathcal{M}_0$ denote the **null model** (without predictor) which simply predicts the sample mean for all observations;
2. for $k = 1, \ldots, p$ (as long as the model can be fit):
    a) fit every model that contains exactly k predictors (there are $\binom{p}{k}$ of them);
    b) pick the model with smallest SSE (largest R^2) and denote it by $\mathcal{M}_k$;
3. select a unique model from $\{\mathcal{M}_0, \ldots, \mathcal{M}_p\}$ using $\text{CV}_{(K)}$, C_p (AIC), BIC, R^2_a, or any other appropriate metric.[46]

BSS is conceptually simple, but with 2^p models to try out, it quickly becomes computationally infeasible for large p ($p > 40$, say). When p is large, the chances of finding a model that performs well according to step 3 but poorly for new data increase, which can lead to **overfitting** and **high-variance** estimates, which were exactly the problems we were trying to avoid in the first place.[47]

20.4.2 Stepwise Selection

Stepwise selection (SS) methods attempt to overcome this challenge by only looking at a restricted set of models. **Forward stepwise selection** (FSS) starts with the null model $\mathcal{M}_0$ and adding predictors one-by-one until it reaches the full model $\mathcal{M}_p$:

1. Let $\mathcal{M}_0$ denote the null model;
2. for $k = 0, \ldots, p-1$ (as long as the model can be fit):
    a) consider the $p - k$ models that add a single predictor to $\mathcal{M}_k$;
    b) pick the model with smallest SSE (largest R^2) and denote it by $\mathcal{M}_{k+1}$;
3. select a unique model from $\{\mathcal{M}_0, \ldots, \mathcal{M}_p\}$ using $\text{CV}_{(K)}$, C_p (AIC), BIC, R^2_a, or any other appropriate metric.

Backward stepwise selection (also BSS, unfortunately) works the other way, starting with the full model $\mathcal{M}_p$ and removing predictors one-by-one until it reaches the null model $\mathcal{M}_0$:

1. Let $\mathcal{M}_p$ denote the full model;
2. for $k = p, \ldots, 1$ (as long as the model can be fit):
    a) consider the k models that remove a single predictor from $\mathcal{M}_k$;

46: We cannot use SSE or R^2 as metrics in this last step, as we would always select $\mathcal{M}_p$ since SSE decreases monotonically with k and R^2 increases monotonically with k. Low SSE/high R^2 are associated with a low training error, whereas the other metrics attempt to say something about the test error, which is what we are after: after all, a model is good if it makes good predictions!

47: Here, we are assuming that all models are OLS models, but subset selection algorithms can be used for other families of supervised learning methods; all that is required are appropriate training error estimates for step 2b and test error estimates for step 3.

b) pick the model with smallest SSE (largest R^2) and denote it by $\mathcal{M}_{k-1}$;

3. select a unique model from $\{\mathcal{M}_0, \ldots, \mathcal{M}_p\}$ using $\text{CV}_{(K)}$, C_p (AIC), BIC, R_a^2, or any other appropriate metric.

The computational advantage of SS over B(est)SS is evident: instead of having to fit 2^p models, SS only requires

$$1 + p + (p-1) + \cdots + 2 + 1 = \frac{p^2 + p + 2}{2}$$

models to be fit to Tr. However, there is no guarantee that the "best" model (among the 2^p BSS models) is found in the reduced set of SS models.

SS can be used in settings where p is too large for BSS to be computationally feasible. Note that for OLS models, backward stepwise selection only works if $p \leq n$ (otherwise OLS might not have a unique parameter solution); if $p > n$, only forward stepwise selection is viable.

Hybrid selection (HS) methods attempt to mimic BSS while keeping model computation in a manageable range, not unlike in SS. More information on this topic is available in [5].

20.4.3 Selecting the Optimal Model

The full model always has largest R^2/smallest SSE.[48] In order to estimate the test error,[49] we can either:

- **adjust** the training error to account for the bias induced by overfitting, or
- **directly** estimate the test error using a validation set or cross-validation.

48: As it is a measure of the training error, and as such, is subject to the overfitting property found in the bias-variance trade-off diagram of Figure 20.5.

49: And thus pick the optimal model in the list $\{\mathcal{M}_0, \ldots, \mathcal{M}_p\}$.

Adjustment Statistics Commonly, we use one of the following adjustment statistics: Mallow's C_p, the Akaike information criterion (AIC), the Bayesian information criteria (BIC), or the adjusted coefficient of determination R_a^2; C_p, AIC, and BIC must be **minimized**, while R_a^2 must be **maximized**.

The adjustment statistics require the following quantities:

- N, the number of observations in Tr;
- p, the number of predictors under consideration;
- $d = p + 2$,
- $\hat{\sigma}^2$, the estimate of $\text{Var}(\varepsilon)$ (irreducible error);
- SSE and SST, the residual and the total sum of squares.

Mallow's C_p statistic is given by

$$C_p = \frac{1}{N}(\text{SSE} + 2d\hat{\sigma}^2) = \text{MSE}_{\text{Tr}} + \underbrace{\frac{2d\hat{\sigma}^2}{N}}_{\text{adjustment}}.$$

As d increases, so does the adjustment term. Note that if $\hat{\sigma}^2$ is an unbiased estimate of $\text{Var}(\varepsilon)$, C_p is an unbiased estimate of MSE_{Te}.

The **Akaike information criterion** (AIC) is given by

$$\text{AIC} = -2 \ln L + \underbrace{2d}_{\text{adjustment}},$$

where L is the maximized value of the likelihood function for the estimated model. If the errors are normally distributed, this requires finding the maximum of

$$\begin{aligned} L &= \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\hat{\sigma}} \exp\left(-\frac{(Y_i - \mathbf{X}_i^\top \boldsymbol{\beta})^2}{2\hat{\sigma}^2}\right) \\ &= \frac{1}{(2\pi)^{N/2}\hat{\sigma}^N} \exp\left(-\frac{1}{2\hat{\sigma}^2} \sum_{i=1}^{N} (Y_i - \mathbf{X}_i^\top \boldsymbol{\beta})^2\right), \end{aligned}$$

or, upon taking the logarithm,

$$\ln L = \text{constant} - \frac{1}{2\hat{\sigma}^2} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2,$$

and so

$$\arg\max_{\boldsymbol{\beta}} \{\ln L(\boldsymbol{\beta})\} = \arg\min_{\boldsymbol{\beta}} \{\|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2\}.$$

However,

$$\begin{aligned} \text{AIC} = -2 \ln L + 2d &= \text{constant} + \frac{1}{\hat{\sigma}^2} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + 2d \\ &= \text{constant} + \frac{\text{SSE}}{\hat{\sigma}^2} + 2d \\ &= \text{constant} + \frac{N}{\hat{\sigma}^2} \cdot \frac{1}{N} \left(\text{SSE} + 2d\hat{\sigma}^2\right) = \text{constant} + \frac{N}{\hat{\sigma}^2} C_p. \end{aligned}$$

Evidently, when the error structure is normal, minimizing AIC is equivalent to minimizing C_p.

The **Bayesian information criterion** uses a different adjustment term:

$$\text{BIC} = \frac{1}{N}(\text{SSE} + d\hat{\sigma}^2 \ln N) = \text{MSE}_{\text{Tr}} + \underbrace{d\hat{\sigma}^2 \frac{\ln N}{N}}_{\text{adjustment}}.$$

This adjustment penalizes models with large number of predictors; minimizing BIC results in selecting models with fewer variables than those obtained by minimizing C_p, in general.

The **adjusted coefficient of determination** R_a^2 is the Ur-example of an adjusted statistic:

$$R_a^2 = 1 - \frac{\text{SSE}/(N - p - 1)}{\text{SST}/(N - 1)} = 1 - (1 - R^2)\frac{N - 1}{N - p - 1}.$$

Maximizing R_a^2 is equivalent to minimizing $\text{SSE}/(N - p - 1)$; note that R_a^2 penalizes models with **unnecessary** variables.[50]

50: Note that in this subsection's formalism, we have $p + 1$ predictors for the linear model: $X_1, \ldots, X + p$ and a constant term X_0.

Validation and Cross-Validation (Reprise) As above, we want to select $\mathcal{M}_{k^*}$ from a sequence of models $\{\mathcal{M}_1, \mathcal{M}_2, \ldots\}$. The procedure is simple: we compute MSE_{Va} on some validation set or $\text{CV}_{(K)}$ for each $\mathcal{M}_k$, and we find the k^* for which the value is **smallest** (see Section 20.3, *Cross-Validation*).

The main advantages of this approach are that:

- there is no need to estimate the irreducible error $\text{Var}(\varepsilon) = \sigma^2$;
- the method produces an estimate for MSE_{Te} "for free," and
- it can be used when the number of parameters is hard to pinpoint (in deep learning networks, for instance).

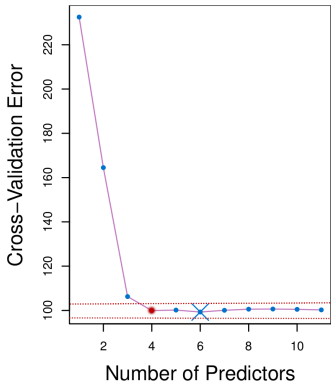
Historically, adjustment approaches have been preferred because cross-validation was computationally demanding, especially when p, n were large, but that is not as much of a problem in modern times.

Consequently, cross-validation is championed as the optimal model selection approach, using the **one standard error rule**: calculate the standard error of $\widehat{\text{MSE}}_{\text{Te}}$ for each model size, and select the **smallest model** for which $\widehat{\text{MSE}}_{\text{Te}}$ is within one standard error from the lowest point on the cross-validation error curve.

Roughly speaking, this is equivalent to **Occam's Razor**[51] on models that have similar predictive power.

51: "When presented with competing hypotheses about the same prediction, one should select the solution with the fewest assumptions."

In the image below (modified from [5]), the lowest point is reached when $p = 6$ (blue "X") and the dashed red lines represent the 1-standard error limits; according to the rule described above, we would select the model with $p = 4$ parameters (red dot).



SS methods are used extensively in practice, but there are serious limitations to this approach:

- all intermediate tests are **biased**, as they are based on the same data;
- R^2_a only takes into account the number of features in the final model, not the degrees of freedom that have been used up during the entire process;
- if the cross-validation error is used, stepwise selection should be repeated for each sub-model.

All in all, SS is a classic example of $p-$hacking: we are getting results without setting hypotheses up first.

Example In spite of the warning mentioned above, it could still be useful to know how to perform stepwise selection. In what follows, we search for the best FSS and BSS linear models to predict the credit card balance for observations contained in the training set Credit.csv.

```
Credit <- read.csv("Credit.csv", stringsAsFactors = TRUE)
str(Credit)
```

```
'data.frame':   400 obs. of  12 variables:
 $ X        : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Income   : num  14.9 106 104.6 148.9 55.9 ...
 $ Limit    : int  3606 6645 7075 9504 4897 8047 3388 7114 ...
 $ Rating   : int  283 483 514 681 357 569 259 512 266 491 ...
 $ Cards    : int  2 3 4 3 2 4 2 2 5 3 ...
 $ Age      : int  34 82 71 36 68 77 37 87 66 41 ...
 $ Education: int  11 15 11 11 16 10 12 9 13 19 ...
 $ Gender   : Factor w/ 2 levels "Female","Male": 2 1 2 1 2 2 ...
 $ Student  : Factor w/ 2 levels "No","Yes": 1 2 1 1 1 1 ...
 $ Married  : Factor w/ 2 levels "No","Yes": 2 2 1 1 2 1 ...
 $ Ethnicity: Factor w/ 3 levels "African American",..: 3 2 2 ...
 $ Balance  : int  333 903 580 964 331 1151 203 872 279 ...
```

We remove the id variable X, and create dummy variables for the categorical levels.

```
Credit <- Credit[,-c(1)]
Credit$Gender.dummy <- ifelse(Credit$Gender == "Female",1,0)
Credit$Student.dummy <- ifelse(Credit$Student == "Yes",1,0)
Credit$Married.dummy <- ifelse(Credit$Married == "Yes",1,0)
Credit$Ethnicity.AA.dummy <- ifelse(Credit$Ethnicity == "African American",1,0)
Credit$Ethnicity.A.dummy <- ifelse(Credit$Ethnicity == "Asian",1,0)
Credit <- Credit[,c(1:6,12:16,11)]
summary(Credit)
```

```
     Income           Limit           Rating          Cards      
 Min.   : 10.35   Min.   :  855   Min.   : 93.0   Min.   :1.000  
 1st Qu.: 21.01   1st Qu.: 3088   1st Qu.:247.2   1st Qu.:2.000  
 Median : 33.12   Median : 4622   Median :344.0   Median :3.000  
 Mean   : 45.22   Mean   : 4736   Mean   :354.9   Mean   :2.958  
 3rd Qu.: 57.47   3rd Qu.: 5873   3rd Qu.:437.2   3rd Qu.:4.000  
 Max.   :186.63   Max.   :13913   Max.   :982.0   Max.   :9.000  

      Age          Education      Gender.dummy    Student.dummy   
 Min.   :23.00   Min.   : 5.00   Min.   :0.0000   Min.   :0.0  
 1st Qu.:41.75   1st Qu.:11.00   1st Qu.:0.0000   1st Qu.:0.0  
 Median :56.00   Median :14.00   Median :1.0000   Median :0.0  
 Mean   :55.67   Mean   :13.45   Mean   :0.5175   Mean   :0.1  
 3rd Qu.:70.00   3rd Qu.:16.00   3rd Qu.:1.0000   3rd Qu.:0.0  
 Max.   :98.00   Max.   :20.00   Max.   :1.0000   Max.   :1.0  
```

```
 Married.dummy    Ethnicity.AA.dummy Ethnicity.A.dummy    Balance
 Min.   :0.0000   Min.   :0.0000     Min.   :0.000     Min.   :   0.00
 1st Qu.:0.0000   1st Qu.:0.0000     1st Qu.:0.000     1st Qu.:  68.75
 Median :1.0000   Median :0.0000     Median :0.000     Median : 459.50
 Mean   :0.6125   Mean   :0.2475     Mean   :0.255     Mean   : 520.01
 3rd Qu.:1.0000   3rd Qu.:0.0000     3rd Qu.:1.000     3rd Qu.: 863.00
 Max.   :1.0000   Max.   :1.0000     Max.   :1.000     Max.   :1999.00
```

We will work with a scaled version of the dataset.

```
Credit.scaled <- scale(Credit)
parameters <- attributes(Credit.scaled)
Credit.scaled <- data.frame(Credit.scaled)
var.names <- colnames(Credit.scaled)
```

We start by implementing step 2 of the FSS algorithm.

```
model <- c()
ind   <- c()

for(i in 1:(ncol(Credit.scaled)-1)){
r2 <- c()
  for(j in setdiff((1:(ncol(Credit.scaled)-1)),c(ind))){
    model <- lm(Balance ~ .,
                data = Credit.scaled[,c(ind,j,12)])
    r2[j] <- summary(model)$r.squared
  }
ind[i] <- which.max(r2)
}

var.names[ind]
```

```
 [1] "Rating"       "Income"            "Student.dummy"
 [4] "Limit"        "Cards"             "Age"
 [7] "Gender.dummy" "Ethnicity.AA.dummy" "Married.dummy"
[10] "Education"    "Ethnicity.A.dummy"
```

The best 1-parameter model $\mathscr{M}_1$ uses `Rating`, the best 2-parameter model $\mathscr{M}_2$ built from $\mathscr{M}_1$ uses `Rating` and `Income`, and so on.

Next, we implement step 3 by computing the adjustment statistics (AIC, BIC, R^2_a) and the cross-validation error (with $K = 5$ folds) for each of $\mathscr{M}_0, \mathscr{M}_1, \ldots$.[52]

52: The latter uses the function `cv.lm()` available in the `lmvar` package in R.

We deal with $\mathscr{M}_0$ first.

```
model <- c()
aic   <- c()
bic   <- c()
r2a   <- c()
cv.m  <- c()
```

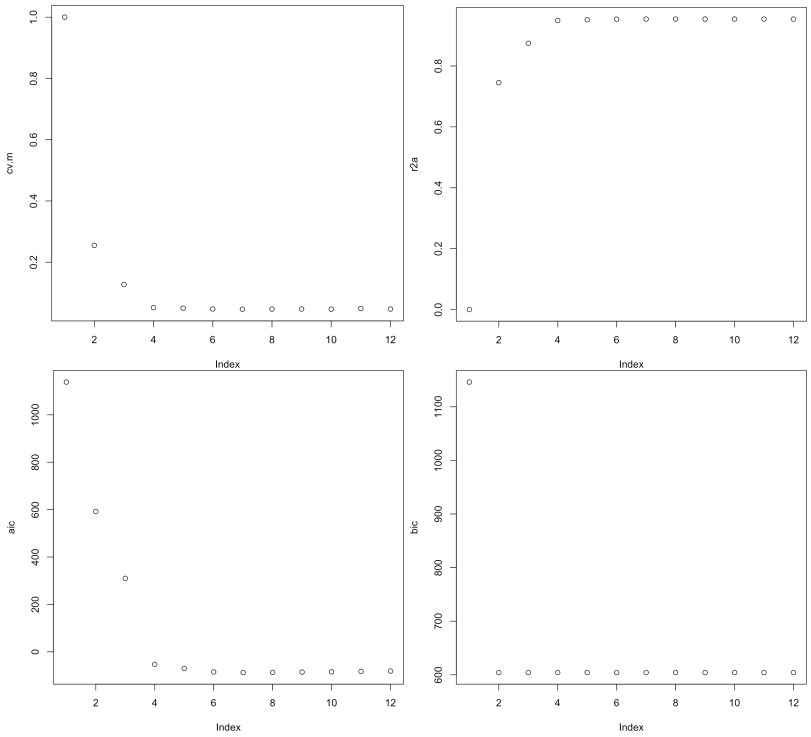
```
model[[1]] <- lm(Balance ~ 1,
                 data=Credit.scaled, y=TRUE, x=TRUE)
cv.m[1]    <- lmvar::cv.lm(model[[1]],k=5)$MSE[[1]]
r2a[1]     <- summary(model[[1]])$adj.r.squared
aic[1]     <- AIC(model[[1]])
bic[1]     <- BIC(model[[1]])
```

The remaining models are similarly handled:

```
for(i in 1:(ncol(Credit.scaled)-1)){
  model[[i+1]] <- lm(Balance ~., data=Credit.scaled[,
            c(ind[c(1:i)],12)], y=TRUE, x=TRUE)
  cv.m[i+1]    <- lmvar::cv.lm(model[[i+1]],k=5)$MSE[[1]]
  r2a[i+1]     <- summary(model[[i+1]])$adj.r.squared
  aic[i+1]     <- AIC(model[[i+1]])
  bic[i+1]     <- BIC(model[[1+1]])
}
```

Let us plot the outcome for each adjustment statistic (and the CV estimation of the test error):

```
plot(cv.m)
plot(r2a)
plot(aic)
plot(bic)
```



The best FSS model using the CV estimate of the test error is:

```
ind.cv  <- which.min(cv.m)
var.names[ind[1:ind.cv]]
```

```
[1] "Rating"        "Income"        "Student.dummy" "Limit"
[5] "Cards"         "Age"
```

The best FSS model using R_a^2 is:

```
ind.r2a <- which.max(r2a)
var.names[ind[1:ind.r2a]]
```

```
[1] "Rating"             "Income"             "Student.dummy"
[4] "Limit"              "Cards"              "Age"
[7] "Gender.dummy"       "Ethnicity.AA.dummy" "Married.dummy"
```

The best FSS model using AIC is:

```
ind.aic <- which.min(aic)
var.names[ind[1:ind.aic]]
```

```
[1] "Rating"        "Income"        "Student.dummy" "Limit"
[5] "Cards"         "Age"           "Gender.dummy"
```

The best FSS model using BIC is:

```
ind.bic <- which.min(bic)
var.names[ind[1:ind.bic]]
```

```
[1] "Rating" "Income"
```

Are there overlaps? The same can be done for BSS, instead:

```
model.BSS <- c()
ind.BSS   <- list()

for(i in 1:(ncol(Credit.scaled)-1)){
  r2 <- c()
  list.of.indices <- combn(1:(ncol(Credit.scaled)-1), i)
  for(j in 1:ncol(list.of.indices)){
    model.BSS <- lm(Balance ~ .,
        data=Credit.scaled[,c(list.of.indices[,j],12)])
    r2[j] <- summary(model.BSS)$r.squared
  }
  ind.BSS[[i]] <- list.of.indices[,which.max(r2)]
}

model.BSS <- c()
aic.BSS   <- c()
bic.BSS   <- c()
r2a.BSS   <- c()
```

```
cv.m.BSS  <- c()

model.BSS[[1]] <- lm(Balance ~ 1, data=Credit.scaled,
                     y=TRUE, x=TRUE)
cv.m.BSS[1] <- lmvar::cv.lm(model.BSS[[1]],
                     k=5)$MSE[[1]]
r2a.BSS[1] <- summary(model.BSS[[1]])$adj.r.squared
aic.BSS[1] <- AIC(model.BSS[[1]])
bic.BSS[1] <- BIC(model.BSS[[1]])

for(i in 1:(ncol(Credit.scaled)-1)){
 model.BSS[[i+1]] <- lm(Balance ~.,
  data=Credit.scaled[,c(ind.BSS[[i]],12)], y=TRUE, x=TRUE)
 cv.m.BSS[i+1] <- lmvar::cv.lm(model.BSS[[i+1]],k=5)$MSE[[1]]
 r2a.BSS[i+1] <- summary(model.BSS[[i+1]])$adj.r.squared
 aic.BSS[i+1] <- AIC(model.BSS[[i+1]])
 bic.BSS[i+1] <- BIC(model.BSS[[1+1]])
}
```

```
ind.cv.BSS  <- which.min(cv.m.BSS)
var.names[ind.BSS[[ind.cv.BSS]]]
ind.r2a.BSS <- which.max(r2a.BSS)
var.names[ind.BSS[[ind.r2a.BSS]]]
ind.aic.BSS <- which.min(aic.BSS)
var.names[ind.BSS[[ind.aic.BSS]]]
ind.bic.BSS <- which.min(bic.BSS)
var.names[ind.BSS[[ind.bic.BSS]]]
```

```
[1] "Income"        "Limit"         "Rating"        "Cards"
[5] "Age"           "Student.dummy"

[1] "Income"             "Limit"              "Rating"
[4] "Cards"              "Age"                "Gender.dummy"
[7] "Student.dummy"      "Married.dummy"      "Ethnicity.AA.dummy"

[1] "Income"        "Limit"         "Rating"        "Cards"
[5] "Age"           "Gender.dummy"  "Student.dummy"

[1] "Income" "Rating"
```

Any surprises?

20.5 Nonlinear Modeling

In practice the linearity assumption is **almost never met** and the regression function

$$y = f(\mathbf{x}) = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}]$$

has to be approximated by some other technique. Or does it?

The linearity assumption is often "good enough" in spite of it not being met, and, coupled with its convenience of use and its multiple extensions, it is rarely a waste of time to give that approach a try.

When heavier machinery is required, it pays to consider the following OLS generalizations, which offer a lot of flexibility without sacrificing ease of interpretability, before jumping to so-called **black box models** (SVM, ANN, ensemble learning, etc.) of Chapter 21:

- curve fitting (polynomial regression, step functions, splines, etc.);
- local regression methods, or
- generalized additive models.

20.5.1 Basis Function Models

If we have reason to suspect that the response Y is not a linear combination of the predictors, we might benefit from using a **derived set of predictors** (see [5, Section 7.3]).

Polynomial Regression We can extend the simple linear regression model $y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, i = 1, \ldots, N$, by allowing for **polynomial basis terms** in the regression function:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_d x_i^d + \varepsilon_i, \quad i = 1, \ldots, N.$$

The regression function is non-linear in terms of the observations x_i, but it is linear in terms of the coefficients β_j.[53] We thus create new variables $X_1 = X$, $X_2 = X^2$, and so on, and estimate the regression function $y = f(\mathbf{x})$ *via* $\hat{f}(\mathbf{x}) = \mathbf{x}^\top \hat{\boldsymbol{\beta}}$, where the coefficients $\hat{\boldsymbol{\beta}}$ are learned using the training set Tr.

53: In terms of $\{x, x^2, \ldots, x^d\}$.

Typically, the coefficient values are of little interest – it is the predictions $\hat{f}(\mathbf{x})$ that are sought.

It is easy to obtain and estimate for $\mathrm{Var}(\hat{f}(\mathbf{x}))$ since $\hat{f}(\mathbf{x})$ is linear in the coefficients $\hat{\beta}_i, i = 0, \ldots, d$:

$$\begin{aligned}\mathrm{Var}(\hat{f}(\mathbf{x})) = \mathrm{Var}(\mathbf{x}^\top \hat{\boldsymbol{\beta}}) &= \sum_{i,j=0}^{d} \mathrm{Cov}(\hat{\beta}_i \tilde{x}_i, \hat{\beta}_j \tilde{x}_j) \\ &= \sum_{i,j=0}^{d} \tilde{x}_i \tilde{x}_j \mathrm{Cov}(\hat{\beta}_i, \hat{\beta}_j) = \mathbf{x}^\top \mathrm{Cov}(\hat{\boldsymbol{\beta}}) \mathbf{x} = \sigma^2 \mathbf{x}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x}.\end{aligned}$$

The estimated variance of the approximation at $\mathbf{x}$ is thus

$$\hat{\mathrm{Var}}(\hat{f}(\mathbf{x})) = \frac{\mathrm{SSRes}}{N-d-1} \mathbf{x}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x} = \frac{\|\mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}}\|_2^2}{N-d-1} \mathbf{x}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x},$$

with $\mathrm{se}(\hat{f}(\mathbf{x})) = \sqrt{\hat{\mathrm{Var}}(\hat{f}(\mathbf{x}))}$, so that

$$\hat{f}(\mathbf{x}) \pm 2 \cdot \mathrm{se}(\hat{f}(\mathbf{x}))$$

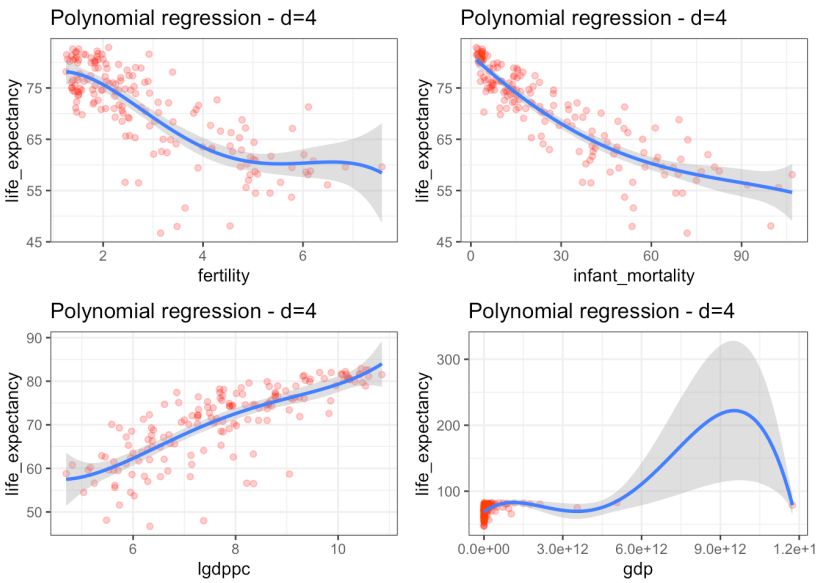
constitutes a 95% C.I. for $\hat{f}(\mathbf{x})$, assuming normality of the error terms.

Gapminder Example The charts below show polynomial regressions ($d = 4$) and confidence intervals for life expectancy against 4 different predictors in the 2011 Gapminder data (assuming that the training set Tr is the entire dataset).[54]

54: In this section, we assume that `ggplot2` and `dplyr` have already been loaded.

```
plot1 <- ggplot(gapminder.2011, aes(x=fertility,
    y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  stat_smooth(method='lm', formula = y~poly(x,4)) +
  ggtitle("Polynomial regression - d=4")

plot2 <- ggplot(gapminder.2011, aes(x=infant_mortality,
    y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  stat_smooth(method='lm', formula = y~poly(x,4)) +
  ggtitle("Polynomial regression - d=4")
```

```
plot3 <- gapminder.2011 |>
  mutate(lgdppc=log(gdp/population)) |>
  ggplot(aes(x=lgdppc, y=life_expectancy)) +
  geom_point(color='red', alpha=0.3) +
  stat_smooth(method='lm', formula = y~poly(x,4)) +
  ggtitle("Polynomial regression - d=4") +
  theme_bw()

plot4 <- ggplot(gapminder.2011, aes(x=gdp,
    y=life_expectancy)) +
  geom_point(color='red', alpha=0.3) +
  stat_smooth(method='lm', formula = y~poly(x,4)) +
  ggtitle("Polynomial regression - d=4") +
  theme_bw()

gridExtra::grid.arrange(plot1, plot2, plot3, plot4, ncol=2)
```



In this example, we picked $d = 4$. How do we select d, in general? We can either pick a reasonable small d (often below 4) or use cross-validation to select a d that minimizes the estimated MSE_{Te}.

Note that it is easy to incorporate more than one predictor and interaction terms into the model.

The nature of polynomials ($|f(\mathbf{x})| \to \infty$ when $\|\mathbf{x}\| \to \infty$) is such that tail behaviour is usually quite horrible (look at the bottom-right example above). Consequently, polynomial regression should be used **very carefully**, staying within the domain and making sure to centre the predictors to reduce variance inflation.

Step Functions Polynomial regression is an attractive approach because of the ease with which we can use the apparatus of OLS, but the elephant in the room is that we are imposing a **global structure** on the non-linear function $y = f(\mathbf{x})$, and that cannot always be justified.

Step functions can be used to keep things "local". Let c_i, $i = 1, \ldots, K$ lie in range(X) and consider the following $K + 1$ new predictors:

$$\begin{aligned} C_0(X) &= \mathcal{I}(X < c_1) \\ C_i(X) &= \mathcal{I}(c_i \le X < c_{i+1}),\ i = 1, \ldots, K-1 \\ C_K(X) &= \mathcal{I}(c_k \le X), \end{aligned}$$

where $\mathcal{I}$ is the **indicator function**

$$\mathcal{I}(\alpha) = \begin{cases} 0, & \alpha \text{ is false} \\ 1, & \alpha \text{ is true} \end{cases}$$

For any X, $C_0(X) + C_1(X) + \cdots + C_K(X) = 1$, since X lies in exactly one of the intervals

$$(-\infty, c_1),\ [c_1, c_2),\ \cdots,\ [c_{K-1}, C_K),\ [C_K, \infty).$$

The **step function regression model** is

$$Y_i = \beta_0 + \beta_1 C_1(X_i) + \cdots + \beta_K C_K(X_i) + \varepsilon_i, \quad i = 1, \ldots, N;$$

it can also be obtained using the OLS framework.[55]

55: Thus a 95% C.I. can be built just as with polynomial and other regressions.

For a given X, at most one of $C_1(X), \ldots, C_K(X)$ is $\neq 0$; thus, when $X < c_1$, $C_j(X) = 0$ for all $j = 1, \ldots, K$, and so

$$\beta_0 = \text{Avg}\{Y \mid X < c_1\}.$$

For $X \in [c_j, c_{j+1})$, $\hat{y} = \beta_0 + \beta_j$, so β_j represents the **average increase in** Y **for** $[c_j, c_{j+1})$ **relative to** $(-\infty, c_1)$.

The only major challenge with step function regression is that there is no easy way to find the number K and select the position of the breakpoints $c_1, \ldots, c_K$, unless there are natural gaps in the predictors. We will discuss a strategy to determine the number and location of knots when we discuss classification and regression trees in Chapter 21.

We did not discuss how step function regression or polynomial regression could be achieved in higher dimensions, but the principle remains the same (except that the number of parameters increases drastically, which can create some overfitting issues).

Gapminder Example The charts below show step function regressions and confidence intervals for life expectancy against 4 different predictors in the 2011 Gapminder data .[56]

56: Assuming that the training set Tr is the entire dataset.

We start by building a $K = 3$ knots step function model for life expectancy against fertility, using the (arbitrary) knot values at 2, 4, and 6:

```
gapminder.2011 <- gapminder.2011 |>
  mutate(fert0=I(fertility<2),
         fert1=I(2<=fertility & fertility<4),
         fert2=I(4<=fertility & fertility<6),
         fert3=I(6<=fertility))
model.sf.1 = lm(life_expectancy ~ fert0 + fert1 + fert2,
                data=gapminder.2011)
summary(model.sf.1)
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-24.0485  -2.8300   0.2515   3.9669  12.1515

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  60.5444     1.9554  30.963  < 2e-16 ***
fert0TRUE    16.8106     2.0969   8.017 2.04e-13 ***
fert1TRUE    10.2040     2.0845   4.895 2.36e-06 ***
fert2TRUE     0.7814     2.2212   0.352    0.725
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.866 on 162 degrees of freedom
Multiple R-squared:  0.5308,    Adjusted R-squared:  0.5221
F-statistic: 61.09 on 3 and 162 DF,  p-value: < 2.2e-16
```

The corresponding step function is defined with:

```
g.1 <- function(x){
    model.sf.1$coefficients[1] +
    model.sf.1$coefficients[2]*I(x<2) +
    model.sf.1$coefficients[3]*I(2<=x & x<4) +
    model.sf.1$coefficients[4]*I(6<=x)
  }
```

We next build a $K = 4$ knots step function model for life expectancy against infant mortality, using the (arbitrary) knot values 10, 20, 40, 70:

```
gapminder.2011 <- gapminder.2011 |>
  mutate(inf0=I(infant_mortality<10),
    inf1=I(10<=infant_mortality & infant_mortality<20),
    inf2=I(20<=infant_mortality & infant_mortality<40),
    inf3=I(40<=infant_mortality & infant_mortality<70),
    inf4=I(70<=infant_mortality))

model.sf.2 = lm(life_expectancy ~ inf0 + inf1 + inf2 +
                inf3, data=gapminder.2011)
summary(model.sf.2)

g.2 <- function(x){
    model.sf.2$coefficients[1] +
    model.sf.2$coefficients[2]*I(x<10) +
    model.sf.2$coefficients[3]*I(10<=x & x<20) +
    model.sf.2$coefficients[4]*I(20<=x & x<40) +
    model.sf.2$coefficients[5]*I(40<=x & x<70)
  }
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-13.9800  -2.3800   0.3725   2.7622   9.3200

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   56.675      1.207  46.939  < 2e-16 ***
inf0TRUE      22.017      1.340  16.437  < 2e-16 ***
inf1TRUE      17.963      1.390  12.928  < 2e-16 ***
inf2TRUE      11.782      1.429   8.247 5.46e-14 ***
inf3TRUE       5.305      1.399   3.791 0.000211 ***
---
Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1

Residual standard error: 4.183 on 161 degrees of freedom
Multiple R-squared:  0.763, Adjusted R-squared:  0.7571
F-statistic: 129.5 on 4 and 161 DF,  p-value: < 2.2e-16
```

We next build a $K = 3$ knots step function model for life expectancy against the log of gdp per capita, using the (arbitrary) knot values at 6, 8, 10:

```
gapminder.2011 <- gapminder.2011 |>
  mutate(lgdppc=log(gdp/population)) |>
  mutate(lgdppc0=I(lgdppc<6),
         lgdppc1=I(6<=lgdppc & lgdppc<8),
         lgdppc2=I(8<=lgdppc & lgdppc<10),
         lgdppc3=I(10<=lgdppc))

model.sf.3 = lm(life_expectancy ~ lgdppc0 + lgdppc1 + lgdppc2,
                data=gapminder.2011)
summary(model.sf.3)
```

```
g.3 <- function(x){
    model.sf.3$coefficients[1] +
    model.sf.3$coefficients[2]*I(x<6) +
    model.sf.3$coefficients[3]*I(6<=x & x<8) +
    model.sf.3$coefficients[4]*I(8<=x & x<10)
  }
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-21.771  -1.831   0.550   3.691   9.789

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   81.100      1.270  63.857  < 2e-16 ***
lgdppc0TRUE  -20.788      1.708 -12.174  < 2e-16 ***
lgdppc1TRUE  -12.629      1.453  -8.692 3.75e-15 ***
lgdppc2TRUE   -6.012      1.509  -3.984 0.000102 ***
---
Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1

Residual standard error: 5.82 on 162 degrees of freedom
Multiple R-squared:  0.5382,    Adjusted R-squared:  0.5296
F-statistic: 62.93 on 3 and 162 DF,  p-value: < 2.2e-16
```

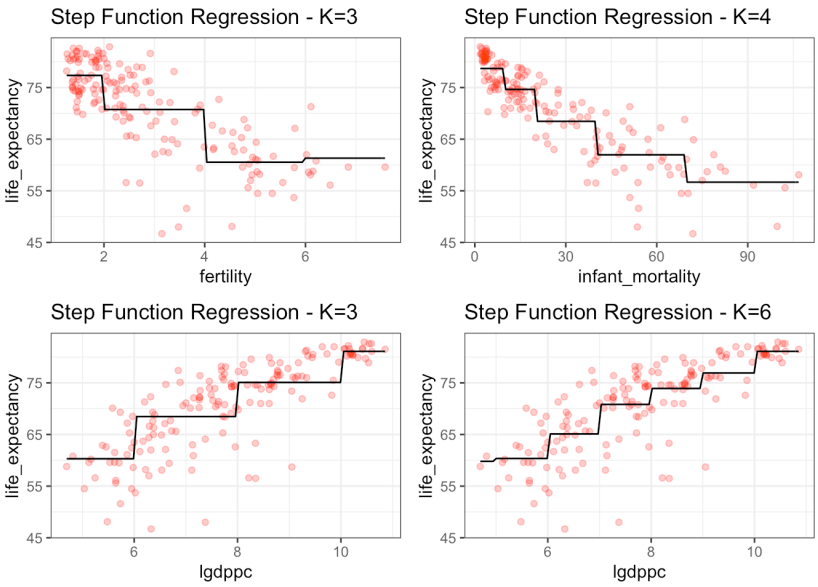
Finally, we build a $K = 6$ knots step function model for life expectancy against the log of gdp per capita, using the (arbitrary) knot values at 5, 6, 7, 8, 9, 10:

```
gapminder.2011 <- gapminder.2011 |>
  mutate(lgdppc=log(gdp/population)) |>
  mutate(lgdppc0=I(lgdppc<5),
         lgdppc1=I(5<=lgdppc & lgdppc<6),
         lgdppc2=I(6<=lgdppc & lgdppc<7),
         lgdppc3=I(7<=lgdppc & lgdppc<8),
         lgdppc4=I(8<=lgdppc & lgdppc<9),
         lgdppc5=I(9<=lgdppc & lgdppc<10),
         lgdppc6=I(10<=lgdppc))

model.sf.4 = lm(life_expectancy ~ lgdppc0 + lgdppc1 +
       lgdppc2 + lgdppc3 + lgdppc4 + lgdppc5,
       data=gapminder.2011)
summary(model.sf.4)

g.4 <- function(x){
    model.sf.4$coefficients[1] +
    model.sf.4$coefficients[2]*I(x<5) +
    model.sf.4$coefficients[3]*I(5<=x & x<6) +
    model.sf.4$coefficients[4]*I(6<=x & x<7) +
    model.sf.4$coefficients[5]*I(7<=x & x<8) +
    model.sf.4$coefficients[6]*I(8<=x & x<9) +
    model.sf.4$coefficients[7]*I(9<=x & x<10)
  }
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-22.8250  -1.3500   0.5964   3.1841  12.2929

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   81.100      1.204  67.367  < 2e-16 ***
lgdppc0TRUE  -21.300      4.082  -5.217 5.59e-07 ***
lgdppc1TRUE  -20.746      1.648 -12.585  < 2e-16 ***
lgdppc2TRUE  -15.993      1.593 -10.042  < 2e-16 ***
lgdppc3TRUE  -10.275      1.487  -6.912 1.10e-10 ***
lgdppc4TRUE   -7.187      1.559  -4.610 8.24e-06 ***
lgdppc5TRUE   -4.190      1.724  -2.431   0.0162 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.517 on 159 degrees of freedom
Multiple R-squared:  0.5927,    Adjusted R-squared:  0.5774
F-statistic: 38.57 on 6 and 159 DF,  p-value: < 2.2e-16
```

The step functions in each of the 4 cases are displayed below:

```
plot1 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  stat_function(fun=g.1) +
  ggtitle("Step Function Regression - K=3")

plot2 <- ggplot(gapminder.2011,aes(x=infant_mortality,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  stat_function(fun=g.2) +
  ggtitle("Step Function Regression - K=4")

plot3 <- ggplot(gapminder.2011,aes(x=lgdppc,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  stat_function(fun=g.3) +
  ggtitle("Step Function Regression - K=3")

plot4 <- ggplot(gapminder.2011,aes(x=lgdppc,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  stat_function(fun=g.4) +
  ggtitle("Step Function Regression - K=6")

gridExtra::grid.arrange(plot1, plot2, plot3, plot4, ncol=2)
```



The step functions do capture the general trends, but how easily interpretable are they?

20.5.2 Splines

We can combine polynomial regression and step functions to obtain a more flexible **curve fitting** approach.

Regression Splines Instead of fitting a polynomial over the entire range of the predictor X, we use different polynomials (of degree up to 3, usually) in regions R_k,[57] such as:

$$Y_i = \begin{cases} \beta_{0,1} + \beta_{1,1} X_i + \beta_{2,1} X_i^2 + \beta_{3,1} X_i^3 + \varepsilon_i, & \text{if } X_i \in R_1 \\ \beta_{0,2} + \beta_{1,2} X_i + \beta_{2,2} X_i^2 + \beta_{3,2} X_i^3 + \delta_i, & \text{if } X_i \in R_2 \end{cases}$$

Various constraints can be imposed on the polynomials:

- none;
- continuity at each region's borders;
- C^1 (continuously differentiable) at each region's borders; etc.

In a sense to be defined shortly, **splines** have the "maximum" amount of continuity. Note that using more regions leads to a more flexible fit.

In what follows, we assume that the domain is split into $K + 1$ regions, bounded by **knots** (there are thus K such knots). If we impose **no restriction** on the functions, we are trying to fit $K + 1$ piecewise cubic functions to the data; each polynomial has 4 parameters to be estimated, leading to $4(K + 1)$ effective parameters.

If we impose a **continuous fit**,[58] we reduce the number of effective parameters. We can also require a **continuously differentiable** fit,[59] further reducing the number of effective parameters.

A **cubic spline** (with only $K + 4$ parameters to fit) is a regression spine which is C^2 on its domain.

57: Defined by **knots** in the 1-dimensional case.

58: The polynomials must agree at the knots.

59: The derivatives must also agree at the knots.

Let ξ be a knot and X be a predictor value. The **positive part** function is defined by

$$w_+ = \begin{cases} w & \text{if } w > 0 \\ 0 & \text{else} \end{cases}$$

Formally, the **linear spline** requires $\xi_1, \ldots, \xi_K$ knots and has $K + 1$ effective parameters. The model can be expressed simply using positive parts:

$$Y_i = \beta_0 + \beta_1 X_i + \beta_2 (X_i - \xi_1)_+ + \cdots + \beta_{K+1}(X_i - \xi_K)_+ + \varepsilon_i;$$

the **cubic spline** is:

$$Y_i = \beta_0 + \beta_1 X_i + \beta_2 X_i^2 + \beta_3 X_i^3 + \beta_4 (X_i - \xi_1)_+^3 + \cdots + \beta_{K+3}(X_i - \xi_K)_+^3 + \varepsilon_i,$$

and the **natural cubic spline** is a cubic spline between ξ_1 and ξ_K, with linear extrapolation beyond ξ_1 and ξ_K; this adds 4 extra constraints to the cubic spline and allows for more knots while keeping the number of effective parameters identical to that of the linear spline.

In all instances, the machinery of OLS remains available: predictions, diagnostics, remedial measures, confidence intervals, and extension to logistic regression, as needed.

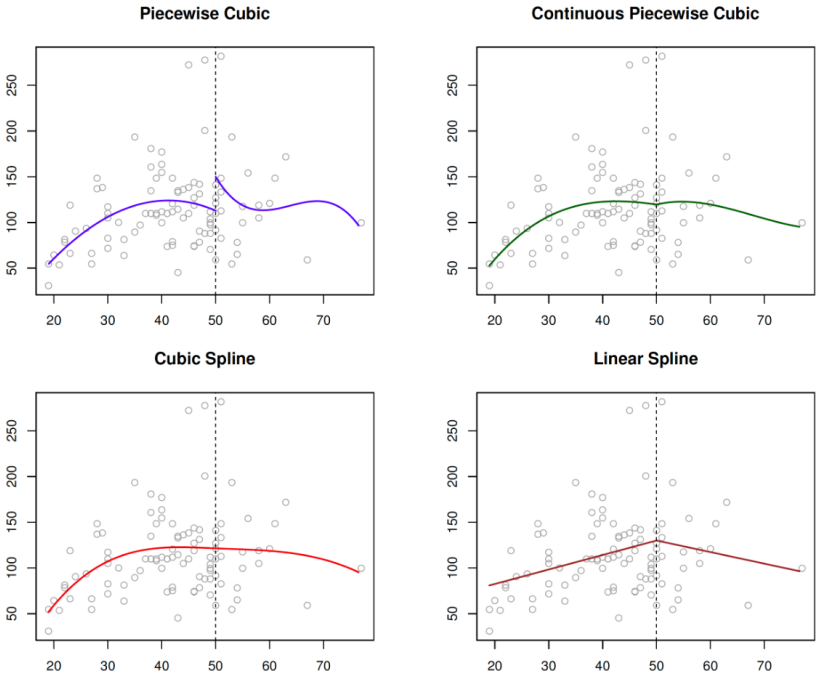


Figure 20.11: Various splines on a 1-dimensional dataset, with a single knot [5].

Gapminder Example The charts below show cubic splines for life expectancy against fertility in the 2011 Gapminder data.[60]

Cubic splines are modeled using the `splines` package `bs()` function. In theory, we place more knots in locations where the spline function is believed to vary more rapidly, and fewer knots where it is more stable.[61]

The syntax for the OLS model formula in `R` follows the form

```
response ~ splines::bs(predictor, df)
```

60: Assuming again that the training set Tr is the entire dataset.

61: In practice, the knots are placed uniformly at **quantiles** of the predictor variable X, based on their number.

where the **degrees of freedom** `df` are linked to the number of parameters to estimate (in the case of cubic spline, df $= K + 3$). We start by building a cubic spline with $K = 0$ knot.[62]

62: So $K + 3 = 3$ degrees of freedom.

```
lm(life_expectancy ~ splines::bs(fertility, df = 3))
```

```
Coefficients:
                    (Intercept)  splines::bs(fertility, df = 3)1
                          79.28                           -11.47
splines::bs(fertility, df = 3)2  splines::bs(fertility, df = 3)3
                         -27.41                           -16.65
```

Here is a cubic spline with $K = 10$ knots, with their locations:[63]

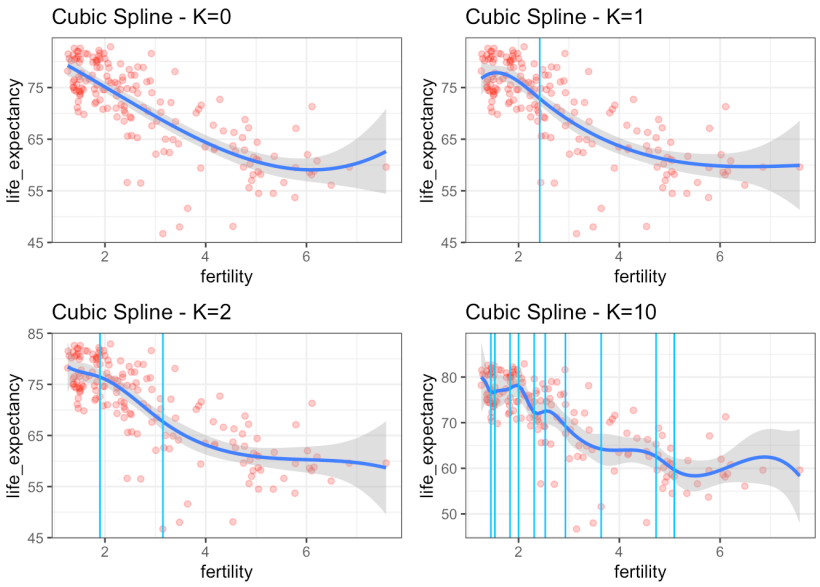
63: We can find the knot locations of a cubic spline with $K = 1, 2$ knots by computing `fm1`, `fm2`, test1, `test2`, `g1`, and `g2` in the same manner (these quantities are required in the display code on the next few pages).

```
fm10 <- lm(life_expectancy ~ splines::bs(fertility, df = 13))
test10 <- eval(attr(fm10$terms, "predvars"))
(g10 <- as.numeric(attr(test10[[2]],"knots")))
```

```
 [1] 1.45 1.53 1.83 2.00 2.31 2.53 2.93 3.64 4.73 5.09
```

We can display cubic splines with $K = 0, 1, 2, 10$ knots as below:

```
plot1 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  geom_smooth(method="lm", formula=y~splines::bs(x,df=3)) +
  ggtitle("Cubic Spline - K=0")

plot2 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  geom_smooth(method="lm", formula=y~splines::bs(x,df=4)) +
  geom_vline(xintercept = g1, colour = "deepskyblue") +
  ggtitle("Cubic Spline - K=1")

plot3 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  geom_smooth(method="lm", formula=y~splines::bs(x,df=5)) +
  geom_vline(xintercept = g2, colour = "deepskyblue") +
  ggtitle("Cubic Spline - K=2")

plot4 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  geom_smooth(method="lm", formula=y~splines::bs(x,df=13)) +
  geom_vline(xintercept = g10, colour = "deepskyblue") +
  ggtitle("Cubic Spline - K=10")

gridExtra::grid.arrange(plot1, plot2, plot3, plot4, ncol=2)
```



Natural cubic splines are modeled using the `splines` package `ns()` function; the knots are, again, placed uniformly at **quantiles** of the predictor variable X, based on their number.[64]

64: The knot locations are thus the same as in the cubic spline case.

The syntax for the OLS model formula in `R` follows the form

```
response ~ splines::ns(predictor, df)
```

where the **degrees of freedom** `df` are linked to the number of parameters to estimate (in the case of natural cubic spline, df $= K + 1$). We start by building a natural cubic spline with $K = 0$ knot.[65]

65: So $K + 1 = 1$ degrees of freedom.

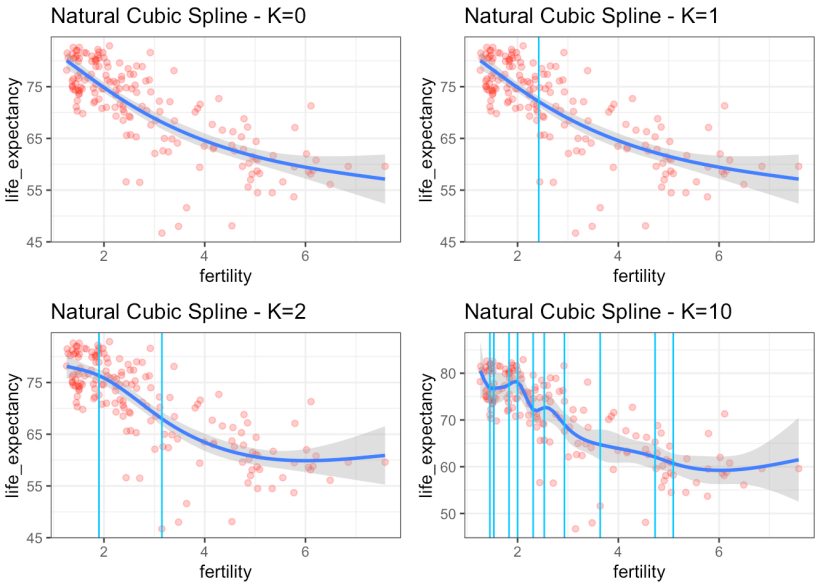
```
lm(life_expectancy ~ splines::ns(fertility, df = 1))
```

```
Coefficients:
                    (Intercept)  splines::ns(fertility, df = 1)
                          78.09                          -34.27
```

The natural cubic splines with $K = 0, 1, 2, 10$ are displayed below:

```
plot1 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  geom_smooth(method="lm", formula=y~splines::ns(x,df=2)) +
  ggtitle("Natural Cubic Spline - K=0")

plot2 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  geom_smooth(method="lm",
   formula=  y ~ splines::ns(x, knots = g1, df = 2)) +
  geom_vline(xintercept = g1, colour = "deepskyblue") +
  ggtitle("Natural Cubic Spline - K=1")
```

```
plot3 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  geom_smooth(method="lm",
   formula=  y ~ splines::ns(x, knots = g2, df = 3)) +
  geom_vline(xintercept = g2, colour = "deepskyblue") +
  ggtitle("Natural Cubic Spline - K=2")

plot4 <- ggplot(gapminder.2011,aes(x=fertility,
       y=life_expectancy)) + theme_bw() +
  geom_point(color='red', alpha=0.3) +
  geom_smooth(method="lm",
   formula=  y ~ splines::ns(x, knots = g10, df = 11)) +
  geom_vline(xintercept = g10, colour = "deepskyblue") +
  ggtitle("Natural Cubic Spline - K=10")

gridExtra::grid.arrange(plot1, plot2, plot3, plot4, ncol=2)
```



Do you notice any difference in the shape of the cubic splines vs. that of the natural cubic splines?[66]

Regression splines often give better results than polynomial regression because they induce **flexibility** *via* a large number of parameters K with low polynomial degree $d \leq 3$, rather than through high d of the latter (and the wild variability that such polynomials have, especially near the boundaries of the predictor's range, as can be observed in the polynomial regression examples above).

66: Cross-validation (again!) can be used to determine the optimal K: compute the estimated error for various K (10-fold CV, say), and pick the K^* that minimizes the error.

Multivariate Adaptive Regression Splines We can reduce the polynomial degree to $d \leq 2$ without losing too much curve fitting accuracy by considering bases consisting of functions of the forms:

$$1, (x - \xi_k)_{\pm}, (x - \xi_{k_1})_{\pm}(x - \xi_{k_2})_{\pm},$$

where $(x-t)_\pm$ is one of the two **hinge functions**:

$$(x-t)_+ = \begin{cases} x-t & \text{if } x > t \\ 0 & \text{else} \end{cases} \qquad (x-t)_- = \begin{cases} t-x & \text{if } x < t \\ 0 & \text{else} \end{cases}$$

$(x-1)_\pm$, $(x-1)_+(x-5)_+$, $(x-1)_+(x-8)_-$ are shown in Figure 20.12.

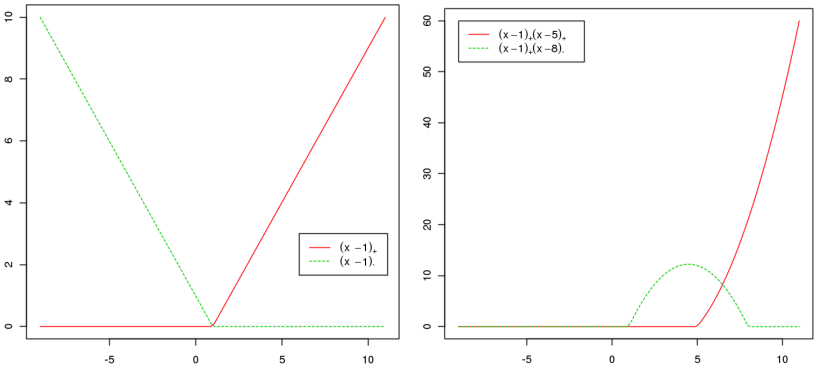


Figure 20.12: A few hinge functions.

A **multivariate adaptive regression spline** (MARS) is expressed as

$$y_i = \sum_{k=1}^{K} \beta_k h_k(x_i) + \varepsilon_i, \quad i = 1, \ldots, N,$$

where h_k is either a constant function, a hinge function, or a product of hinge functions.

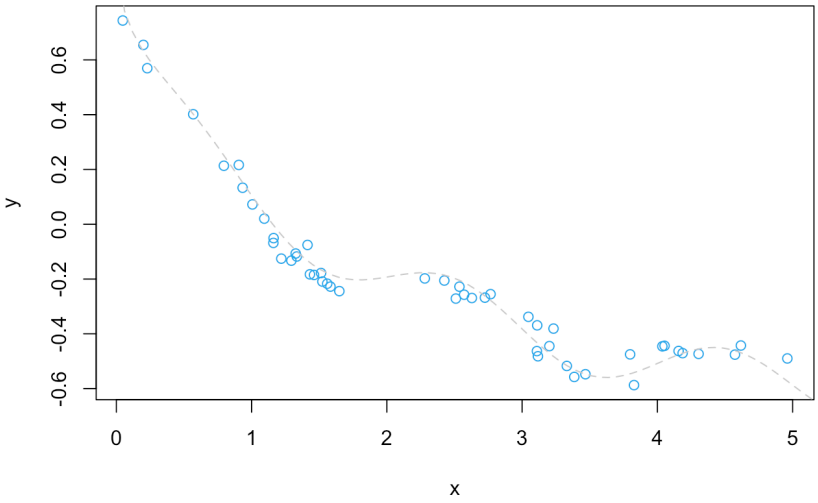
MARS adds terms to its model in an iterative fashioin; once a stopping criterion is met, unwanted terms are removed. The model growth's parallels the growth of tree-based models, which we will discuss in Chapter 21, and it has the same advantage that the knots are selected automatically.

Artificial Dataset Example Let us take a look at a synthetic dataset, based off of:

$$y = f(x) = \frac{\sin(\pi x)}{10} - \sqrt{x} + \exp(x/10) + \varepsilon,$$

where $\varepsilon \sim \mathcal{N}(0, 0.04^2)$.

```
set.seed(1234)
fx=function(x){
  sin(pi*x)/10-sqrt(x)+exp(x/10)
  }

x=sort(runif(50, 0, 5))
noise=rnorm(50, 0, 0.04)
y=fx(x)+noise

plot(x, y, col=4)
x.vec=seq(0,6, length.out=100)
lines(x.vec, fx(x.vec), col="grey", lty=2)
```



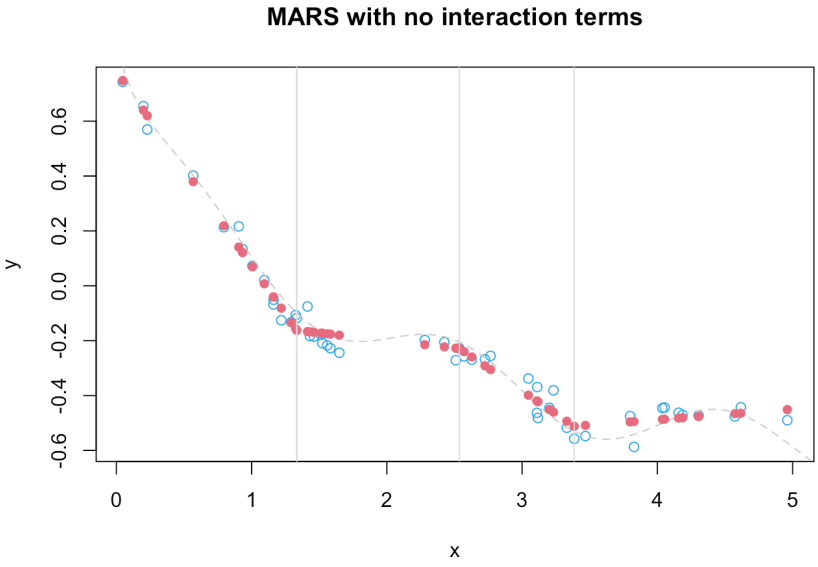
We can fit the data using package `mda`'s `mars()` function, in R.[67] Let us use only functions of degree 1 (linear functions and linear hinges, but no interaction terms) for the time being:

```
MARS.1 = mda::mars(x, y, degree=1)
```

The output is rather lengthy and is suppressed for readability.[68]

Let's see how good a job MARS did:

```
plot(x, y, col=4, main="MARS with no interaction terms")
x.vec=seq(0,6, length.out=100)
lines(x.vec, fx(x.vec), col="grey", lty=2)
points(x, MARS.1$fitted.values, col=2, pch=16)
abline(v = MARS.1$cuts[MARS.1$selected.terms[-1]],
       col = "light grey")
```



Not bad, all things considered.

67: This is a licensed implementation of MARS. There is another implementation in the `earth` package: **enhanced adaptive regression through hinges**, or EARTH.

68:

- `$call` provides the model;
- `$fitted.values` contains the estimated values $\hat{y}_i$;
- `$residuals` contain the residuals $\hat{y}_i - y_i$, and
- `$x` gives the hinge functions used in the final model.

The EARTH output is identical, and would be obtained thus:

```
EARTH.1 = earth::earth(x, y, degree=1)
summary(EARTH.1)
```

```
             coefficients
(Intercept)   -0.16254461
h(1.3341-x)    0.70785002
h(x-1.3341)   -0.05502561
h(x-2.53653)  -0.27853205
h(x-3.38547)   0.37209809

Selected 5 of 6 terms, and 1 of 1 predictors
Termination condition: RSq changed by less than 0.001 at 6 terms
Importance: x
Number of terms at each degree of interaction: 1 4 (additive model)
GCV 0.002341396    RSS 0.07871774    GRSq 0.9759754    RSq 0.9831798
```

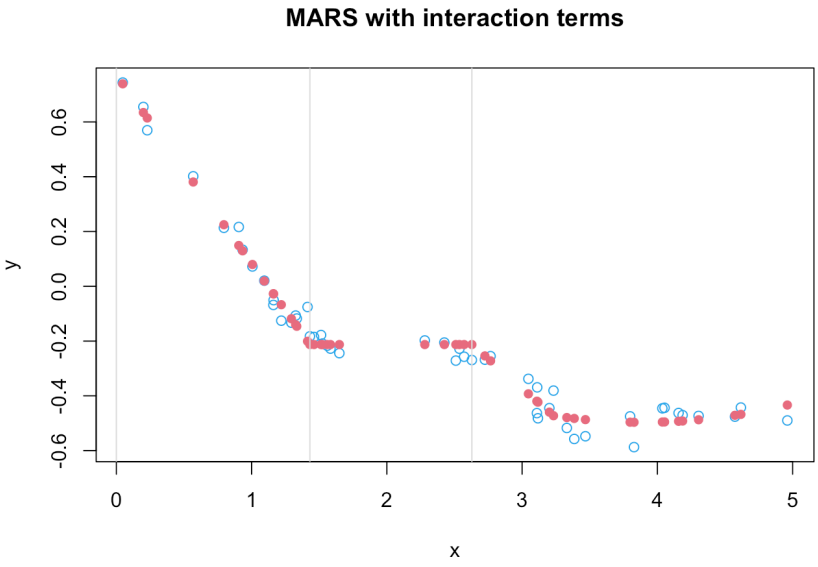
What about interaction terms? In order for MARS or EARTH to consider such terms, we must first provide a second predictor.

```
xnew = x*x
data = data.frame(x,xnew,y)
EARTH.2 = earth::earth(y ~ x + xnew , data=data, degree=2)
summary(EARTH.2)
```

```
               coefficients
(Intercept)     -0.21273261
h(1.43112-x)     0.68806424
h(x-2.62849)    -0.43057541
h(xnew-10.446)   0.05531043

Selected 4 of 6 terms, and 2 of 2 predictors
Termination condition: RSq changed by less than 0.001 at 6 terms
Importance: x, xnew
Number of terms at each degree of interaction: 1 3 (additive model)
GCV 0.00273995    RSS 0.09437758    GRSq 0.971886    RSq 0.9798336
```

What does the plot look like? Can you spot the non-linear components?

```
plot(x, y, col=4, main="MARS with interaction terms")
x.vec=seq(0,6, length.out=100)
points(x, EARTH.2$fitted.values, col=2, pch=16)
abline(v = EARTH.2$cuts[EARTH.2$selected.terms[-1]],
       col = "light grey")
EARTH.2$cuts
```



Housing Dataset Example In this section, we analyze a housing dataset related to house selling prices in Ames, Iowa (VE_Housing.csv, modified from [1]). We start by reading in the data:

```
dat.Housing=read.csv("VE_Housing.csv", header=TRUE,
                     stringsAsFactors = TRUE)
dim(dat.Housing)
```

```
[1] 1460   81
```

Next, we count the number of missing values for each variable, excluding those variables with complete rows.

```
missing = attributes(which(apply(is.na(dat.Housing), 2,
          sum)>0))$names
apply(is.na(dat.Housing[,missing]), 2, sum)
```

```
 LotFrontage        Alley   MasVnrType   MasVnrArea     BsmtQual     BsmtCond
         259         1369            8            8           37           37
BsmtExposure BsmtFinType1 BsmtFinType2   Electrical  FireplaceQu   GarageType
          38           37           38            1          690           81
 GarageYrBlt GarageFinish   GarageQual   GarageCond       PoolQC        Fence
          81           81           81           81         1453         1179
 MiscFeature
        1406
```

The housing dataset thus consists of $n = 1460$ observations with $p = 79$ predictors. There are two other variables: `Id` and `SalePrice`, representing the index variable and the response variable, respectively.[69] Furtheremore, the variables

- `LotFrontage`
- `Alley`

69: Use `colnames()` or `str()` to list all the variables.

- FireplaceQu
- PoolQC
- Fence, and
- MiscFeature

all have anywhere from 259 to 1406 missing observations. The proportions of missing values in these variables are probably too high for imputation (see Chapter 15 for details), so we elect to remove them from further analyses.

Note that the remaining major missing variables are all related to *Garage* and *Basement*, with corresponding variables missing for the same houses. Given that there are other variables associated with these, we suspect these variables will not play a crucial role in model building, and we also elect to remove them from the analyses.

For the remaining three variables with missing values (MasVnrType, MasVnrArea, and Electrical), the number of missing observations are so small that we could easily

- impute these values, or
- perform list-wise deletion.

For the purposes of this example, we will select the latter options and delete all columns with missing values.

```
dat.Housing.new = dat.Housing[,
       !colnames(dat.Housing)%in%missing]
dim(dat.Housing.new)
```

```
[1] 1460   62
```

We also remove the index variable ID:

```
dat.Housing.new = subset(dat.Housing.new, select = -c(Id))
```

In order to evaluate the effectiveness of the eventual model (i.e., to have good predictive power without overfitting the data), we split the Housing dataset into training and testing sets. The model is then developed using the training set (i.e., optimized using a subset of data), and then later tested for its prediction power using the testing set.

We select roughly 80% of the observations (1160) for the training set:

```
set.seed(1234) # for replicability
n.train=1160
ind.train=sample(1:nrow(dat.Housing.new), n.train)
```

The training and testing sets are thus:

```
dat.train=dat.Housing.new[ind.train,]
dat.test=dat.Housing.new[-ind.train,]
```

We train EARTH (with interactions) on the training data:

```
EARTH.3 <- earth::earth(SalePrice~., data=dat.train,
                        degree=2)
summary(EARTH.3)
```

```
                                      coefficients
(Intercept)                              317.95604
Exterior1stBrkFace                        18.17930
FoundationPConc                           34.63490
h(14442-LotArea)                          -0.00198
h(LotArea-14442)                           0.00048
...
h(7-OverallCond) * h(316-WoodDeckSF)      -0.01602
h(2005-YearBuilt) * h(1056-BsmtFinSF1)     0.00030
h(YearBuilt-2005) * h(1056-BsmtFinSF1)    -0.00928

Selected 36 of 39 terms, and 19 of 188 predictors
Termination condition: RSq changed by less than 0.001 at 39 terms
Importance: OverallQual, GrLivArea, YearBuilt, SaleTypeWD, BsmtFinSF1, ...
Number of terms at each degree of interaction: 1 18 17
GCV 396.2527    RSS 392191.8    GRSq 0.9377484    RSq 0.9467931
```

We now predict `SalePrice` on the testing data:

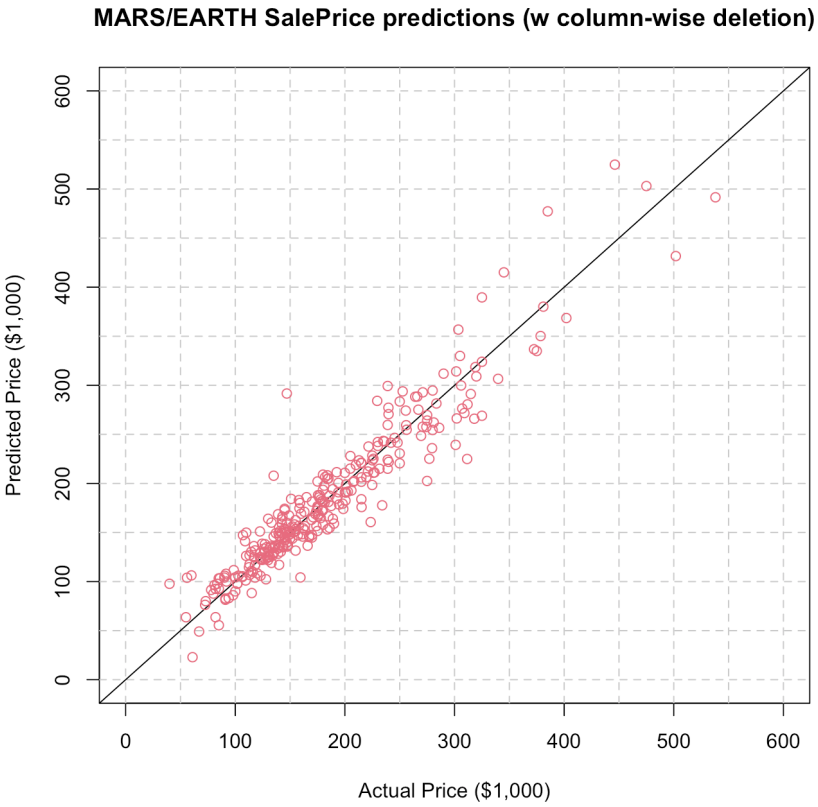
```
yhat.EARTH.3 = predict(EARTH.3, dat.test)
```

We can evaluate the quality of the predictions on the testing set either by computing MSE_{Te} directly (≈ 628), but this value is more or less useless on its own.

We get a better sense for the quality of the prediction on the testing set by comparing the actual `SalePrice` values to the EARTH predicted `SalePrice` values:

```
xlimit = ylimit = c(0,600)
plot(NA, col=2, xlim=xlimit, ylim=ylimit,
     ylab="Predicted Price ($1,000)",
     xlab="Actual Price ($1,000)",
     main="MARS/EARTH SalePrice predictions
           (w column-wise deletion)")
abline(h=seq(0,600, length.out=13), lty=2, col="grey",
       v=seq(0,600, length.out=13))
abline(a=0, b=1)
points(dat.test$SalePrice, yhat.EARTH.3, col=2)
```

(see plot on the next page) What do you think? Is the model likely to prove useful?



Smoothing Splines Given a training set Tr with N observations, we have seen that **regression splines** use the following approach:

1. identify K knots $\xi_1, \ldots, \xi_K$;
2. produce some basis functions $\{b_1(x), \ldots, b_K(x)\}$, and
3. use OLS to estimate the coefficients of

$$Y_i = \beta_0 + \beta_1 b_1(X_i) + \cdots + \beta_K b_k(X_i) + \varepsilon_i, \quad i = 1, \ldots, N.$$

But we can use another mathematical approach in order to produce a spline. In general, we need to find a function g that provides a good fit for the available data; in other words, we are looking for a g for which

$$\text{SSE} = \sum_{i=1}^{N} (Y_i - g(X_i))^2 \quad \text{is "small".}$$

But we also need g to be constrained, otherwise any smooth function interpolating (X_y, Y_i), $i = 1, \ldots, N$ would yield SSE $= 0$, at the cost of **severe overfitting** and loss of **interpretability**, as in Figure 20.13. The flip side is that too many constraints can result in the data being **underfit**.

The **smoothing spline** approach seeks to solve the following problem:

$$g_\lambda = \arg\min_h \left\{ \underbrace{\sum_{i=1}^{N} (Y_i - h(X_i))^2}_{\text{SSE loss}} + \lambda \underbrace{\int_{\Omega(X)} [h''(t)]^2 \, dt}_{\text{penalty term}} \right\},$$

where $\lambda \geq 0$ is a **tuning parameter** and $\Omega(X)$ represents the range of the predictor X.

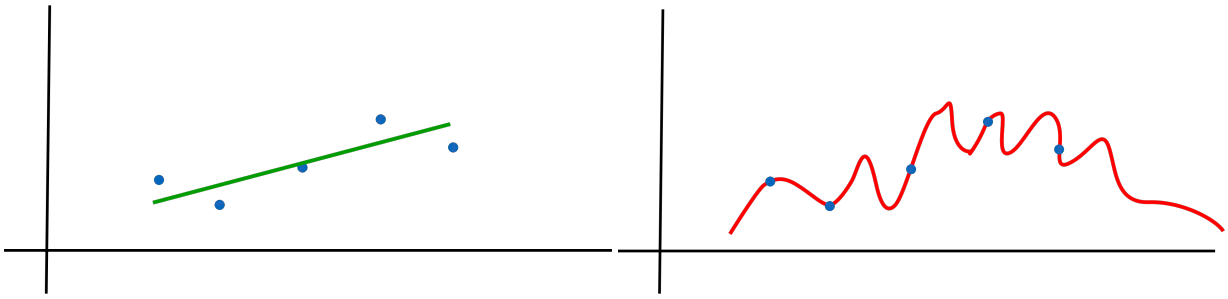


Figure 20.13: A spline with too few constraint overfits the data (right).

The **penalty term** measures the roughness of the spline function h; if h is quite "wiggly", the penalty will be (relatively) large, and **vice-versa** (and similarly for g).[70]

70: If h represents a straight line, say, the penalty term would be zero.

When $\lambda \to 0$, the penalty term has little effect, so we would expect g_λ to be "jumpy" in such cases and that it would interpolate the observations exactly, leading to overfitting.

When $\lambda \to \infty$, the penalty term dominates and g_λ is a function for which $\int [g''_\lambda(t)]^2 dt \to 0$ over $\Omega(X)$, so $g_\lambda \to$ linear OLS solution over $\Omega(X)$, leading to underfitting.

As we have seen over and over again, the tuning parameter λ controls the bias-variance trade-off, expressed, in this case, as a battle between rigidity and model complexity.

The **optimal smoothing spline** g_λ is a natural cubic spline with a knot at every data point $\xi_i = x_i$, $i = 1, \ldots, N$, with continuous 0th, 1st, 2nd derivatives throughout the range $\Omega(X) = [\min \xi_i, \max \xi_i]$, and is linear outside $\Omega(X)$, but, importantly, **it is not the one that would be obtained from building a regression spline**, as it also depends on the turning parameter λ.

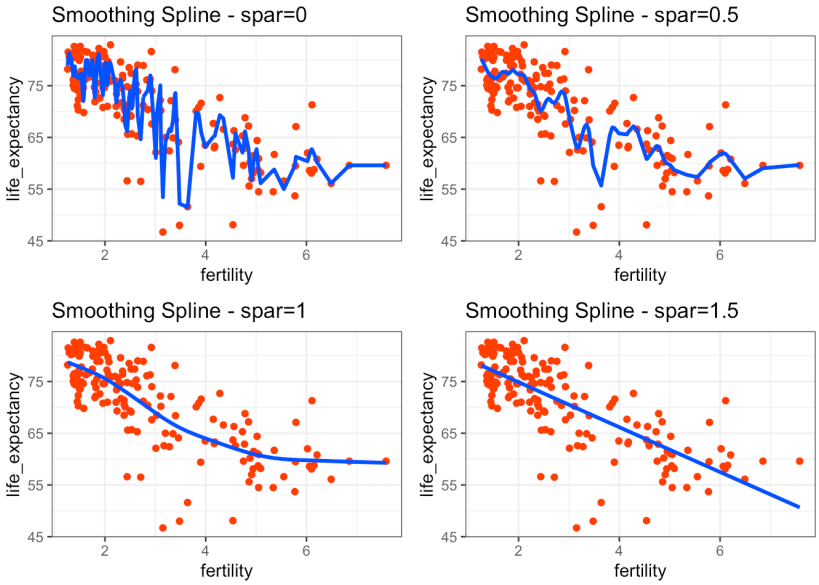
What is the best choice for λ? At first glance, this would seem to be another job for cross-validation, but there is another option: we can specify the smoothing spline through the **effective degrees of freedom**, which decrease from N to 2 as λ goes from 0 to ∞ (note, however, that R's `smooth.spline()` uses a different parameterization).

Gapminder Example The charts below show the smoothing spline for life expectancy against fertility in the 2011 Gapminder data, for 4 different smoothing parameter values, using `stats`'s `smooth.spline()` function. Note that the entire set is used as training data.

```
x=gapminder.2011$fertility
y=gapminder.2011$life_expectancy

ss00 = stats::smooth.spline(x, y, spar=0)
ss05 = stats::smooth.spline(x, y, spar=0.5)
ss10 = stats::smooth.spline(x, y, spar=1)
ss15 = stats::smooth.spline(x, y, spar=1.5)
```

In order to be able to display the smoothing splines over the datapoints, we use the `broom::augment()` function, which provide the value of the spline at the various fertility values in the dataset.

```
plot1 <- ggplot(broom::augment(ss00, gapminder.2011),
                aes(x=fertility, y=life_expectancy)) +
  geom_point(colour="red") + theme_bw() +
  geom_line(aes(y = .fitted), colour="blue", size=1.1) +
  ggtitle("Smoothing Spline - spar=0")

plot2 <- ggplot(broom::augment(ss05, gapminder.2011),
                aes(x=fertility, y=life_expectancy)) +
  geom_point(colour="red") + theme_bw() +
  geom_line(aes(y = .fitted), colour="blue", size=1.1) +
  ggtitle("Smoothing Spline - spar=0.5")

plot3 <- ggplot(broom::augment(ss10, gapminder.2011),
                aes(x=fertility, y=life_expectancy)) +
  geom_point(colour="red") + theme_bw() +
  geom_line(aes(y = .fitted), colour="blue", size=1.1) +
  ggtitle("Smoothing Spline - spar=1")

plot4 <- ggplot(broom::augment(ss15, gapminder.2011),
                aes(x=fertility, y=life_expectancy)) +
  geom_point(colour="red") + theme_bw() +
  geom_line(aes(y = .fitted),  colour="blue", size=1.1) +
  ggtitle("Smoothing Spline - spar=1.5")

gridExtra::grid.arrange(plot1, plot2, plot3, plot4, ncol=2)
```



Note the evolution of a flexible but highly non-interpretable model (the wiggly curve associated to `spar=0`) into a rigid but highly interpretable model (the line associated to `spar=1.5`) as the `spar` values increase.

20.5.3 Generalized Additive Models

While polynomial regression and splines can be applied to predictor sets, they are best-suited to predicting a response Y on the basis of a **single predictor** X (the model complexity increases quickly if more than one predictor is present).

Generalized additive models (GAM) allow for flexible non-linearities in several variables while retaining the **additive structure** of linear models:

$$y_i = \beta_0 + f_1(x_{i,1}) + \cdots + f_p(x_{i,p}) + \varepsilon_i, \quad i = 1, \ldots, N$$

where each of the f_j can be derived using any of the methods previously discussed; if

$$\begin{aligned} f_1(x_i) &= \beta_{1,1} b_{1,1}(x_{i,1}) + \cdots + \beta_{1,L_1} b_{1,L_1}(x_{i,1}) \\ &\vdots \\ f_p(x_i) &= \beta_{p,1} b_{p,1}(x_{i,p}) + \cdots + \beta_{p,L_p} b_{p,L_p}(x_{i,p}), \end{aligned}$$

say, we would fit the data using OLS (but this cannot be done if one of the components is a smoothing spline, for instance, or if it is non-linear in some other way).

In practice, using natural cubic splines for the quantitative components seem to work as well as smoothing spline, when it comes to making predictions.[71]

71: GAM can also be used for classification via log-odds:

$$\ln\left(\frac{p_1(\mathbf{x})}{1 - p_1(\mathbf{x})}\right) = \beta_0 + f_1(x_1) + \cdots + f_p(x_p).$$

GAM are implemented in `R` using the `mgcv::gam()` function; a typical call might look like:

```
mgcv::gam(y ~ s(x1,df=5) +
              lo(x2,spar=0.5) +
              bs(x3,df=4) +
              ns(x4,df=5):ns(x5,df=5) +
              x6, data=dat)
```

which would indicate that the contribution of:

- X_1 is given by smoothing spline with 5 degrees of freedom,
- X_2 is given by a local regression with `spar`=0.5,
- X_3 is given by a cubic spline with 4 degrees of freedom,
- the fourth component is an interaction term based on natural splines for X_4 and X_5 (each with 5 degrees of freedom), and
- X_6 is directly added to the model.

GAM provide a useful compromise between linear models and fully non-parametric models.

Advantages:

- GAM can fit a non-linear f_j to each predictor X_j, so that they could capture trends that linear regression would miss;
- GAM can reduce the number of data transformations to try out manually on each predictor X_j;

- non-linear fits may improve accuracy of predictions for the response Y;
- GAM are useful for inference due to their additivity – the effect of X_j on Y (while keeping other predictors fixed) can be analyzed separately;
- the overall smoothness of the model can be summarized *via* effective degrees of freedom/parameters.

Disadvantages:

- GAM still suffer from the curse of dimensionality;
- GAM are restricted to additive models – interaction terms can be added manually by introducing new predictors $X_j \times X_k$, as can interaction functions $f_{j,k}(X_j, X_k)$ (using local regression or MARS, say), but they quickly get out of hand (due to *Curse of Dimensionality* issues).

Gapminder Example The charts below show the individual contributions of fertility, infant mortality, GDP, and continental membership to life expectancy in the 2011 Gapminder data.[72]

72: Using the entire set as training data.

```
library(mgcv)
b <- gam(gapminder.2011$life_expectancy ~
         s(gapminder.2011$fertility) +
         s(gapminder.2011$infant_mortality) +
         s(gapminder.2011$gdp) +
         gapminder.2011$continent)
summary(b)
```

```
Family: gaussian
Link function: identity

Formula:
gapminder.2011$life_expectancy ~ s(gapminder.2011$fertility) +
    s(gapminder.2011$infant_mortality) + s(gapminder.2011$gdp) +
    gapminder.2011$continent

Parametric coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)        68.1186     0.7470  91.190  < 2e-16 ***
continentAmericas   4.4787     1.1161   4.013 9.30e-05 ***
continentAsia       4.7110     0.9993   4.714 5.35e-06 ***
continentEurope     3.4179     1.3209   2.588   0.0106 *
continentOceania   -0.2891     1.3798  -0.210   0.8343
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
                                    edf Ref.df      F p-value
s(gapminder.2011$fertility)       1.000  1.000  4.474   0.036 *
s(gapminder.2011$infant_mortality) 3.027  3.800 40.541  <2e-16 ***
s(gapminder.2011$gdp)             1.478  1.779  0.367   0.575
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.828   Deviance explained = 83.8%
GCV = 13.199  Scale est. = 12.363     n = 166
```

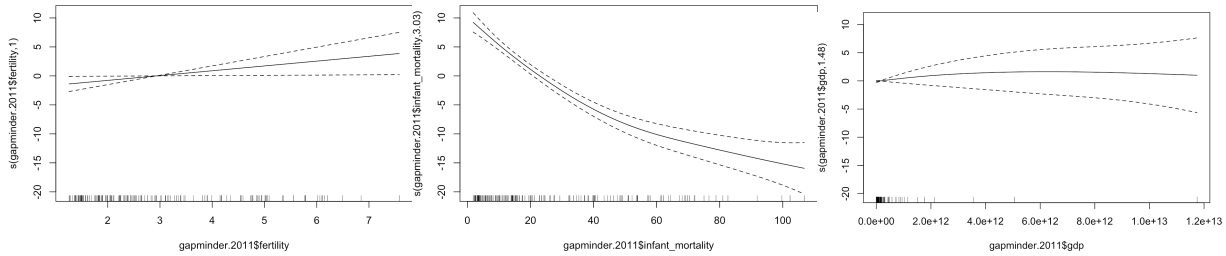
We see in the outcome that the intercept is $\beta_0 = 68.1186$ and that

$$\beta_{\text{continent}} = \begin{cases} 0 & \text{Africa} \\ 4.4787 & \text{America} \\ 4.7110 & \text{Asia} \\ 3.4179 & \text{Europe} \\ -0.2891 & \text{Oceania} \end{cases}$$

so that predictions take the form

$$\begin{aligned} \text{life expectancy} \approx \beta_0 &+ f_1(\text{fertility}) + f_2(\text{infant mortality}) \\ &+ f_3(\text{gdp}) + \beta_{\text{continent}}. \end{aligned}$$

```
plot.gam(b)
```



For instance, the life expectancy for an American country with fertility= 3, infant mortality= 1, GDP= 6×10^{12} would be approximately

$$68.1 + 0 + 10 + 2 + 4.5 = 84.6.$$

Take the time to read the `mgcv` and the `gam` documentation to better understand how these work in practice (in particular, how to make predictions on test/new observations).

20.6 Example: Algae Blooms

We continue the algae blooms analysis started in Section 15.7 (based on a case study by L.Torgo [11]). The objective is to predict various algae levels in water samples; we continue the analysis with the data frame `algae_blooms.sna2`.

20.6.1 Value Estimation Modeling

For **supervised learning** tasks, the *bias-variance* trade-off means that we need to set aside a **testing set** on which the models (which were learned on the **training set**) are evaluated.

There are no hard-and-fast rules to determine how to split the data; if the signal is very strong, a small training set should capture its important features, but we do not typically know how strong the signal is before we start the modeling process. On the other hand, if the training set is too large, we run the risk of overfitting the data. Weighing both of these considerations, we elect to use a 65%/35% training/testing split.

The training data should also be **representative**, to avoid injecting biases in the model (in case the data was provided according to some systematic but unknown rule).

There are numerous ways to do this,[73] but we can do so using a simple random sample of 218 observations.[74]

To avoid issues related to replicability, we use a single training set.[75]

73: See Chapter 10, *Survey Sampling Methods*, for instance.

74: We could also have stratified according to season, size, etc.

75: The code that would allow for a different random sample every time the code is run has been commented out in the following code box.

```
# ind <- sample(1:dim(algae_blooms.sna2)[1], 218)
ind <- 1:218
algae.train <- algae_blooms.sna2[ind,] # training set
algae.test <- algae_blooms.sna2[-ind,] # testing set
set.seed(0)  # for replicability
```

Generalized Linear Model We implement a linear model to predict `a2` (to pick but one of the response variables) against all the predictor variables, but only using the training set.[76]

76: Before getting too excited about using various machine learning methods, it is worth seeing what the traditional approaches yield.

```
linear.model.a2 <- lm(a2 ~ season + size + speed + mxPH +
        mnO2 + Cl + NO3 + NH4 + oPO4 + PO4 + Chla,
        data=algae.train)
```

A summary of the results can be given by calling the `summary` method on the resulting object.

```
summary(linear.model.a2)
```

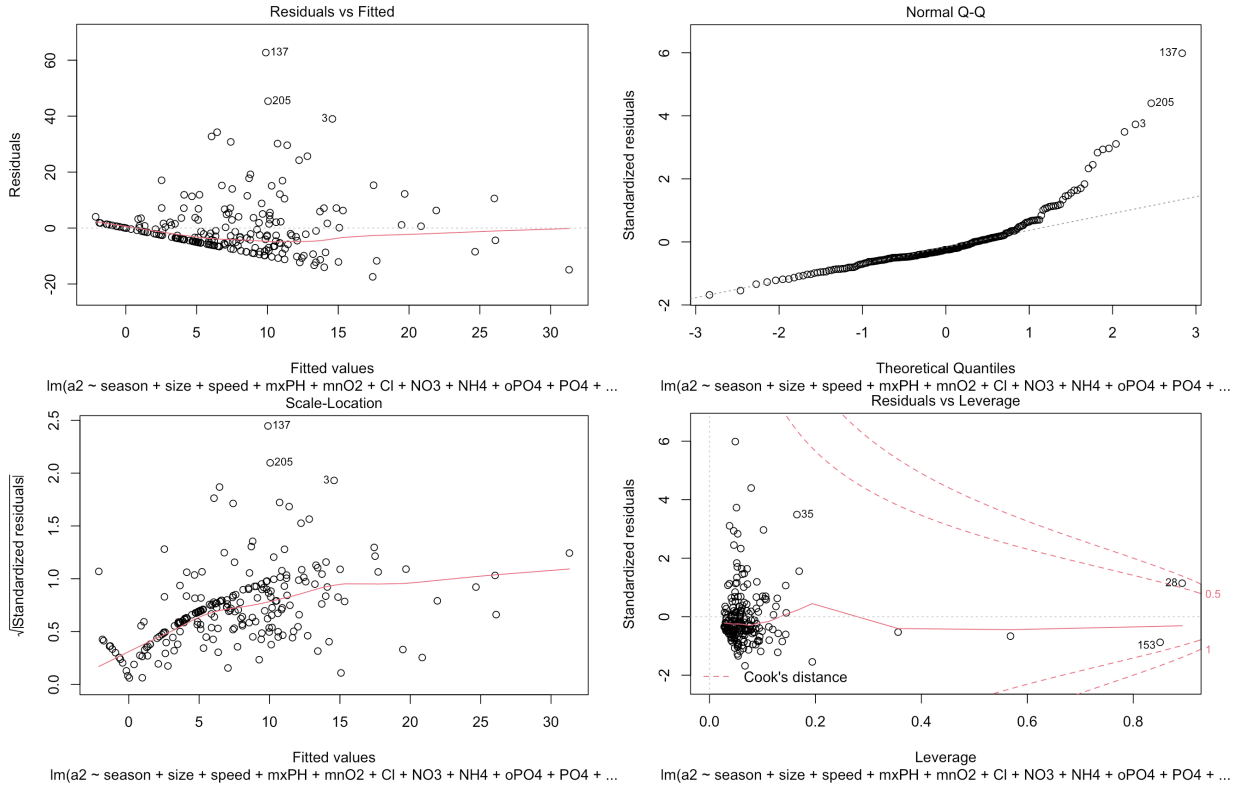
```
Residuals:
    Min      1Q  Median      3Q     Max
-17.436  -5.281  -2.613   2.026  62.712

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)  -3.083e+01  1.257e+01  -2.452 0.015056 *
seasonsummer -1.166e-01  2.112e+00  -0.055 0.956035
seasonautumn  1.071e+00  2.370e+00   0.452 0.651934
seasonwinter -1.451e+00  2.000e+00  -0.726 0.468935
sizemedium   -2.628e+00  1.895e+00  -1.387 0.166896
sizelarge    -3.210e+00  2.412e+00  -1.331 0.184767
speedmedium   3.887e+00  2.485e+00   1.564 0.119325
speedhigh    -1.104e+00  2.772e+00  -0.398 0.690751
mxPH          4.859e+00  1.559e+00   3.117 0.002092 **
mnO2         -1.841e-01  3.924e-01  -0.469 0.639474
Cl           -7.432e-03  2.006e-02  -0.371 0.711351
```

```
NO3           2.132e-01  3.028e-01   0.704 0.482249
NH4          -5.979e-04  5.355e-04  -1.117 0.265510
oPO4          2.290e-03  9.876e-03   0.232 0.816875
PO4          -1.559e-03  5.936e-03  -0.263 0.793090
Chla          1.652e-01  4.614e-02   3.579 0.000432 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.74 on 202 degrees of freedom
Multiple R-squared:  0.206, Adjusted R-squared:  0.147
F-statistic: 3.493 on 15 and 202 DF,  p-value: 2.498e-05
```

We see that the adjusted R^2 coefficient is fairly small, which is not ideal. Furthermore, the residuals should have a mean of 0 and be "small", which is not quite what we are seeing here; the $F-$statistic seems to indicate that there is some (linear) dependence on the predictor variables.

We can get a better handle on the regression diagnostics by calling the `plot()` method on the object.

```
plot(linear.model.a2)
```



All in all, the linear model is ... not great. The significance of some of the coefficients is questionable, however, and we might wonder what effect their inclusion might have.

```
anova(linear.model.a2)
```

```
Analysis of Variance Table
Response: a2
           Df  Sum Sq Mean Sq F value    Pr(>F)
season      3   112.3   37.42  0.3243 0.8078029
size        2   436.0  217.99  1.8892 0.1538604
speed       2  1552.8  776.42  6.7287 0.0014825 **
mxPH        1  2223.5 2223.54 19.2698 1.829e-05 ***
mnO2        1     0.5    0.54  0.0047 0.9455025
Cl          1     0.3    0.33  0.0029 0.9572795
NO3         1    43.9   43.91  0.3806 0.5380001
NH4         1   193.8  193.82  1.6797 0.1964428
oPO4        1     0.1    0.09  0.0008 0.9775762
PO4         1     4.8    4.82  0.0417 0.8383141
Chla        1  1478.2 1478.18 12.8103 0.0004316 ***
Residuals 202 23308.8  115.39
---
Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
```

We might be interested in the results of a linear regression with the `NH4` predictor removed, say.

```
linear.model.a2.mod <- update(linear.model.a2, . ~ . -NH4)
summary(linear.model.a2.mod)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-16.801  -5.500  -2.647   2.504  63.259

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -30.996221  12.580354  -2.464 0.014577 *
seasonsummer  -0.107996   2.113697  -0.051 0.959301
seasonautumn   0.806683   2.359593   0.342 0.732799
seasonwinter  -1.397244   2.000275  -0.699 0.485648
sizemedium    -2.378831   1.882645  -1.264 0.207838
sizelarge     -3.086404   2.411377  -1.280 0.202029
speedmedium    3.637403   2.476278   1.469 0.143408
speedhigh     -1.382060   2.762133  -0.500 0.617364
mxPH           4.821140   1.559264   3.092 0.002268 **
mnO2          -0.074216   0.380118  -0.195 0.845398
Cl            -0.001602   0.019376  -0.083 0.934181
NO3           -0.013968   0.224437  -0.062 0.950437
oPO4          -0.001285   0.009348  -0.137 0.890775
PO4           -0.001518   0.005940  -0.256 0.798576
Chla           0.165865   0.046166   3.593 0.000411 ***
---
Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1

Residual standard error: 10.75 on 203 degrees of freedom
Multiple R-squared:  0.2011,    Adjusted R-squared:  0.146
F-statistic: 3.649 on 14 and 203 DF,  p-value: 2.009e-05
```

The fit is not that much better, but an ANOVA on the 2 suggested models shows that we are at least ≈ 88% certain that the models are different.

```
anova(linear.model.a2,linear.model.a2.mod)
```

```
Analysis of Variance Table
Model 1: a2 ~ season + size + speed + mxPH + mnO2 + Cl + NO3 + NH4 +
    oPO4 + PO4 + Chla
Model 2: a2 ~ season + size + speed + mxPH + mnO2 + Cl + NO3 +
    oPO4 + PO4 + Chla
  Res.Df   RSS Df Sum of Sq      F Pr(>F)
1    202 23309
2    203 23453 -1   -143.86 1.2467 0.2655
```

The `step()` function uses AIC to perform a **model search** (using **backward elimination**). The "best" linear model for `a2` is thus:

```
final.linear.model.a2 <- step(linear.model.a2)
summary(final.linear.model.a2)
```

```
Start:  AIC=1050.52
a2 ~ season + size + speed + mxPH + mnO2 + Cl + NO3 + NH4 + oPO4 + PO4 + Chla

         Df Sum of Sq   RSS    AIC
- season  3    157.44 23466 1046.0
- oPO4    1      6.20 23315 1048.6
- PO4     1      7.96 23317 1048.6
- Cl      1     15.85 23325 1048.7
- mnO2    1     25.40 23334 1048.8
- NO3     1     57.19 23366 1049.0
- size    2    282.28 23591 1049.1
- NH4     1    143.86 23453 1049.9
<none>                23309 1050.5
- speed   2    967.47 24276 1055.4
- mxPH    1   1121.22 24430 1058.8
- Chla    1   1478.18 24787 1061.9

Step:  AIC=1045.98
a2 ~ size + speed + mxPH + mnO2 + Cl + NO3 + NH4 + oPO4 + PO4 + Chla

        Df Sum of Sq   RSS    AIC
- oPO4   1      2.54 23469 1044.0
- PO4    1      4.10 23470 1044.0
- mnO2   1      6.61 23473 1044.0
- Cl     1     15.59 23482 1044.1
- size   2    257.60 23724 1044.4
- NO3    1     47.04 23513 1044.4
- NH4    1    114.06 23580 1045.0
<none>                23466 1046.0
- speed  2   1035.56 24502 1051.4
- mxPH   1   1052.01 24518 1053.5
- Chla   1   1477.06 24943 1057.3
```

```
Step:  AIC=1044.01
a2 ~ size + speed + mxPH + mnO2 + Cl + NO3 + NH4 + PO4 + Chla

        Df Sum of Sq   RSS    AIC
- PO4    1      1.62 23470 1042.0
- mnO2   1      7.17 23476 1042.1
- Cl     1     14.19 23483 1042.1
- NO3    1     44.93 23514 1042.4
- size   2    266.73 23736 1042.5
- NH4    1    114.91 23584 1043.1
<none>               23469 1044.0
- speed  2   1050.55 24519 1049.5
- mxPH   1   1099.78 24569 1052.0
- Chla   1   1480.47 24949 1055.3

Step:  AIC=1042.02
a2 ~ size + speed + mxPH + mnO2 + Cl + NO3 + NH4 + Chla

        Df Sum of Sq   RSS    AIC
- mnO2   1      6.59 23477 1040.1
- Cl     1     17.42 23488 1040.2
- size   2    265.19 23736 1040.5
- NO3    1     51.04 23521 1040.5
- NH4    1    140.72 23611 1041.3
<none>               23470 1042.0
- speed  2   1050.42 24521 1047.6
- mxPH   1   1105.21 24576 1050.0
- Chla   1   1482.34 24953 1053.4

Step:  AIC=1040.08
a2 ~ size + speed + mxPH + Cl + NO3 + NH4 + Chla

        Df Sum of Sq   RSS    AIC
- Cl     1     13.41 23490 1038.2
- size   2    260.65 23738 1038.5
- NO3    1     44.48 23522 1038.5
- NH4    1    135.66 23613 1039.3
<none>               23477 1040.1
- speed  2   1121.64 24599 1046.3
- mxPH   1   1103.17 24580 1048.1
- Chla   1   1492.55 24970 1051.5

Step:  AIC=1038.21
a2 ~ size + speed + mxPH + NO3 + NH4 + Chla

        Df Sum of Sq   RSS    AIC
- NO3    1     36.13 23526 1036.5
- size   2    275.91 23766 1036.8
- NH4    1    128.31 23619 1037.4
<none>               23490 1038.2
- speed  2   1172.78 24663 1044.8
- mxPH   1   1089.85 24580 1046.1
- Chla   1   1490.94 24981 1049.6
```

```
Step:  AIC=1036.54
a2 ~ size + speed + mxPH + NH4 + Chla

        Df Sum of Sq   RSS    AIC
- size   2    244.91 23771 1034.8
- NH4    1     93.48 23620 1035.4
<none>               23526 1036.5
- speed  2   1164.36 24691 1043.1
- mxPH   1   1053.88 24580 1044.1
- Chla   1   1611.04 25138 1049.0

Step:  AIC=1034.8
a2 ~ speed + mxPH + NH4 + Chla

        Df Sum of Sq   RSS    AIC
- NH4    1     82.62 23854 1033.6
<none>               23771 1034.8
- mxPH   1    850.56 24622 1040.5
- speed  2   1085.45 24857 1040.5
- Chla   1   1540.50 25312 1046.5

Step:  AIC=1033.56
a2 ~ speed + mxPH + Chla

        Df Sum of Sq   RSS    AIC
<none>               23854 1033.6
- speed  2   1021.27 24875 1038.7
- mxPH   1    928.72 24783 1039.9
- Chla   1   1479.59 25334 1044.7

Call:
lm(formula = a2 ~ speed + mxPH + Chla, data = algae.train)

Residuals:
    Min      1Q  Median      3Q     Max
-16.195  -6.008  -2.530   2.024  63.589

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -27.13270   11.07921  -2.449 0.015134 *
speedmedium   4.17176    2.34330   1.780 0.076453 .
speedhigh    -0.32929    2.41899  -0.136 0.891850
mxPH          3.89794    1.35358   2.880 0.004387 **
Chla          0.15945    0.04387   3.635 0.000349 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.58 on 213 degrees of freedom
Multiple R-squared:  0.1874,    Adjusted R-squared:  0.1721
F-statistic: 12.28 on 4 and 213 DF,  p-value: 5.289e-09
```

It is still not a great fit (the adjusted R^2 is quite small); we conclude that the linear model is not ideal to predict `a2`.

```
anova(final.linear.model.a2)
```

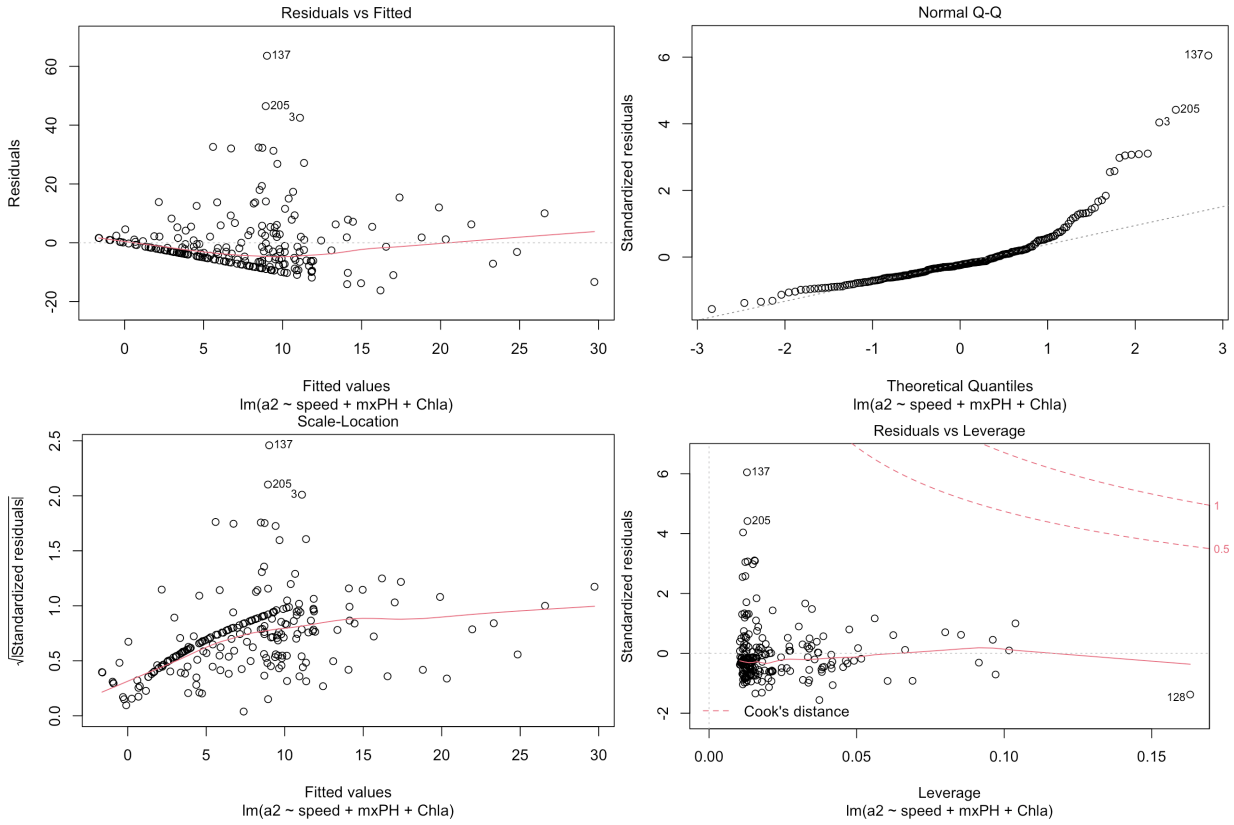
```
Analysis of Variance Table
Response: a2
           Df  Sum Sq Mean Sq F value    Pr(>F)
speed       2  1994.8  997.42  8.9063 0.0001929 ***
mxPH        1  2026.6 2026.63 18.0964 3.145e-05 ***
Chla        1  1479.6 1479.59 13.2117 0.0003488 ***
Residuals 213 23854.1  111.99
---
Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
```

In spite of the final model’s poor quality, it is significantly different from the full model.

```
anova(linear.model.a2,final.linear.model.a2)
```

```
Model 1: a2 ~ season + size + speed + mxPH + mnO2 + Cl +
    NO3 + NH4 + oPO4 + PO4 + Chla
Model 2: a2 ~ speed + mxPH + Chla
  Res.Df   RSS  Df Sum of Sq      F Pr(>F)
1    202 23309
2    213 23854 -11   -545.26 0.4296 0.9416
```

```
plot(final.linear.model.a2)
```



Regression Tree Model An alternative to regression is the use of **regression trees**, implemented in the function `rpart()`.[77]

77: Its syntax is similar to `lm()`.

```
regression.tree.a2 <-rpart::rpart(a2 ~ season + size +
    speed + mxPH + mnO2 + Cl + NO3 + NH4 + oPO4 + PO4 +
    Chla, data=algae.train)
```

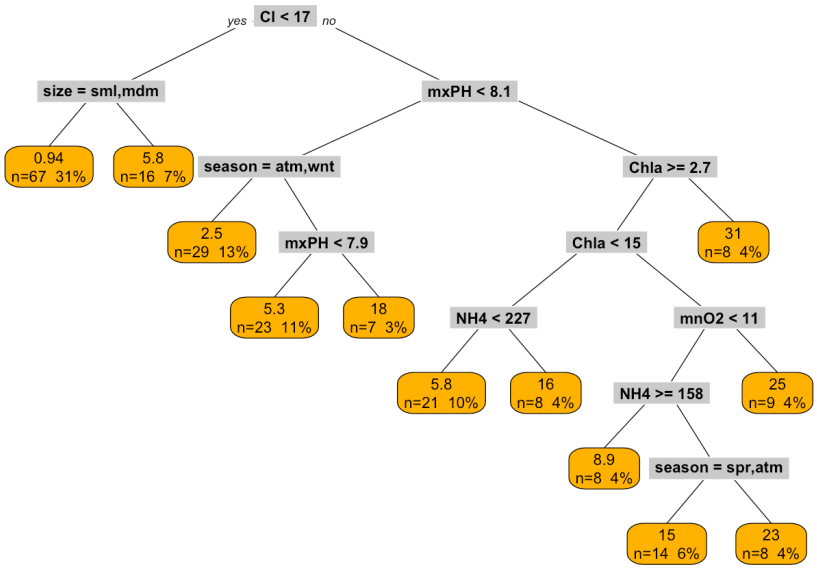
The outcome can be displayed by calling the object directly.

```
regression.tree.a2
```

```
node), split, n, deviance, yval
      * denotes terminal node

  1) root 218 29355.1300  7.6366970
    2) Cl< 16.6875 83  1193.6400  1.8891570
      4) size=small,medium 67   398.6457  0.9447761 *
      5) size=large 16   485.0194  5.8437500 *
    3) Cl>=16.6875 135 23733.9200 11.1703700
      6) mxPH< 8.065 59  3831.8290  5.3864410
       12) season=autumn,winter 29   561.8414  2.5172410 *
       13) season=spring,summer 30  2800.4720  8.1600000
         26) mxPH< 7.9375 23   889.9730  5.3173910 *
         27) mxPH>=7.9375 7  1114.0000 17.5000000 *
      7) mxPH>=8.065 76 16396.0400 15.6605300
       14) Chla>=2.65 68  9694.0890 13.8544100
         28) Chla< 14.8875 29  2747.5810  8.7172410
           56) NH4< 226.875 21   558.4257  5.7857140 *
           57) NH4>=226.875 8  1534.9490 16.4125000 *
         29) Chla>=14.8875 39  5612.0940 17.6743600
           58) mnO2< 11.05 30  3139.0940 15.4233300
            116) NH4>=158.409 8   577.1000  8.9000000 *
            117) NH4< 158.409 22  2097.7700 17.7954500
              234) season=spring,autumn 14   674.7521 14.6642900 *
              235) season=summer,winter 8  1045.5550 23.2750000 *
           59) mnO2>=11.05 9  1814.2760 25.1777800 *
       15) Chla< 2.65 8  4594.6690 31.0125000 *
```

The tree structure can be hard to determine when there is a large number of nodes; we can improve on the visuals by using the R library `rpart.plot`.

```
rpart.plot::prp(regression.tree.a2, extra=101,
                box.col="orange", split.box.col="gray")
```



Details on the regression tree can be obtained by calling the `summary()` method on the object.

```
summary(regression.tree.a2)
```

```
           CP nsplit rel error    xerror      xstd
1  0.15082765      0 1.0000000 1.0059069 0.1990911
2  0.11943572      1 0.8491724 0.9492709 0.1815913
3  0.07178590      2 0.7297366 0.8655117 0.1688012
4  0.04545758      3 0.6579507 0.9007445 0.1699016
5  0.02243987      4 0.6124932 0.9597254 0.1737117
6  0.02228595      5 0.5900533 0.9472199 0.1658890
7  0.02156378      6 0.5677673 0.9472199 0.1658890
8  0.01581407      8 0.5246398 0.9287217 0.1629262
9  0.01285848      9 0.5088257 0.9255472 0.1613858
10 0.01055949     10 0.4959672 0.9320459 0.1622581
11 0.01000000     11 0.4854077 0.9389544 0.1625727

Variable importance
  Chla    NH4     Cl   mxPH   oPO4    PO4    NO3  speed
    19     14     14     13     11      9      6      5
  mnO2 season   size
     4      3      2
```

Note that `rpart()` grows a tree on the training data until one of the following criterion is met: - decrease in deviance goes below a certain threshold (`cp`) - number of samples in a node is below some other threshold (`minsplit`) - depth of the tree crosses yet another threshold (`maxdepth`)

The library also implements a **pruning method** based on *cost complexity*: finding the value of `cp` which best balances **predictive accuracy** and **tree size**.[78]

78: We will revisit these notions in Section 21.4.1, *Tree-Based Methods*.

```
rpart::printcp(regression.tree.a2)
```

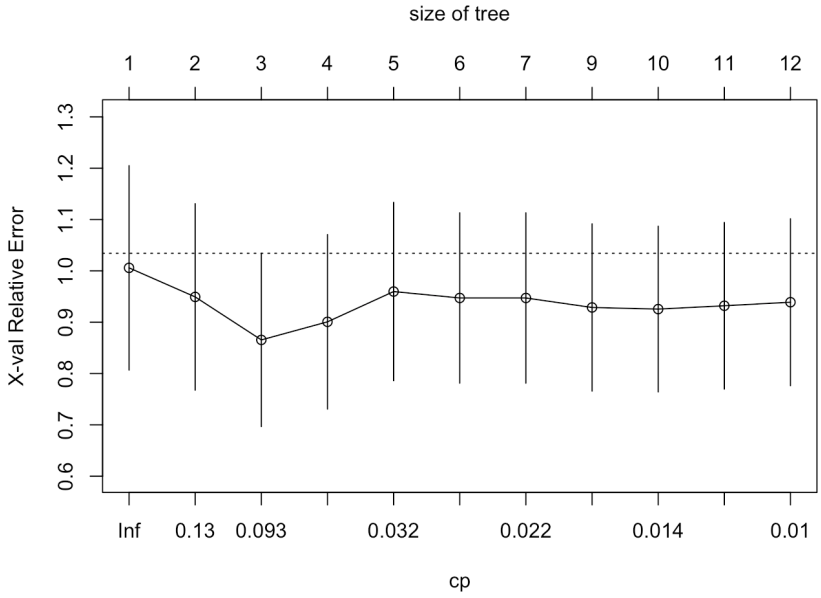
```
Variables actually used in tree construction:
[1] Chla   Cl     mnO2   mxPH   NH4    season size

Root node error: 29355/218 = 134.66

n= 218

         CP nsplit rel error  xerror    xstd
1  0.150828      0   1.00000 1.00591 0.19909
2  0.119436      1   0.84917 0.94927 0.18159
3  0.071786      2   0.72974 0.86551 0.16880
4  0.045458      3   0.65795 0.90074 0.16990
5  0.022440      4   0.61249 0.95973 0.17371
6  0.022286      5   0.59005 0.94722 0.16589
7  0.021564      6   0.56777 0.94722 0.16589
8  0.015814      8   0.52464 0.92872 0.16293
9  0.012858      9   0.50883 0.92555 0.16139
10 0.010559     10   0.49597 0.93205 0.16226
11 0.010000     11   0.48541 0.93895 0.16257
```

The tree returned by `rpart()` is the final one (`cp`= 0.01 is the default value); it requires 11 decision tests, and has a relative error of 0.485. Internally, `rpart()` uses 10-fold cross-validation to estimate that the tree has an average relative error of 0.98 ± 0.18.[79]

79: These values might change when from one run to the next due to the stochastic nature of the internal cross-validation routines.

In this framework, the optimal tree minimizes the value of `xerror`. Alternatively, one could use the **1 − SE** rule to find the minimal `xerror` + `xstd` tree.

```
rpart::plotcp(regression.tree.a2)
```

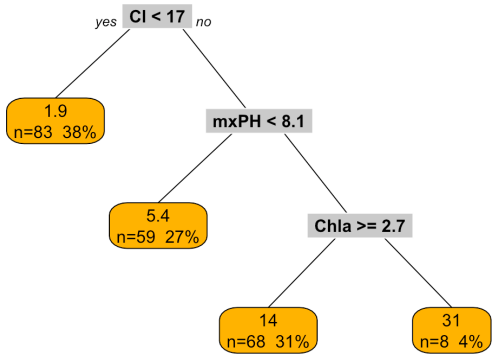


The scree plot above suggests that cp= 0.08 (that value may change when you run yours due to the stochastic nature of the internal cross-validation algorithm) is a special value for tree growth, so we could prune the tree using that specific value.

```
(regression.tree.a2.mod <- rpart::prune(
                              regression.tree.a2,cp=0.05))
```

```
node), split, n, deviance, yval
      * denotes terminal node

 1) root 218 29355.130  7.636697
   2) Cl< 16.6875 83  1193.640  1.889157 *
   3) Cl>=16.6875 135 23733.920 11.170370
     6) mxPH< 8.065 59  3831.829  5.386441 *
     7) mxPH>=8.065 76 16396.040 15.660530
      14) Chla>=2.65 68  9694.089 13.854410 *
      15) Chla< 2.65 8  4594.669 31.012500 *
```

```
rpart.plot::prp(regression.tree.a2.mod, extra=101,
                box.col="orange", split.box.col="gray")
```



The entire process is automated in the wrapper method rpartXse() provided with the DMwR library;[80] we (abitrarily) use se= 0.2.

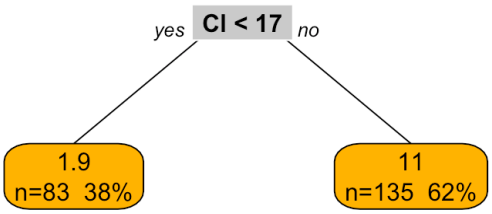
80: This library had to be installed from source files as it was not available on the Comprehensive R Archive Network as of January 2023.

```
library(DMwR)
(regression.tree.a2.final <- DMwR::rpartXse(a2 ~ season +
    size + speed + mxPH + mnO2 + Cl + NO3 + NH4 + oPO4 +
    PO4 + Chla, data=algae.train, se=0.2))
summary(regression.tree.a2.final)
```

```
node), split, n, deviance, yval
      * denotes terminal node

1) root 218 29355.13  7.636697
  2) Cl< 16.6875 83  1193.64  1.889157 *
  3) Cl>=16.6875 135 23733.92 11.170370 *
Call:
```

```
rpart(formula = form, data = data, cp = cp, minsplit = minsplit)
  n= 218

        CP nsplit rel error    xerror      xstd
1 0.1508276      0 1.0000000 1.0130822 0.2001495
2 0.1194357      1 0.8491724 0.9320224 0.1752140

Variable importance
   Cl   PO4  oPO4  Chla   NH4 speed
   28    17    16    14    14    12

Node number 1: 218 observations,    complexity param=0.1508276
  mean=7.636697, MSE=134.6565
  left son=2 (83 obs) right son=3 (135 obs)
  Primary splits:
      Cl   < 16.6875 to the left,  improve=0.15082760, (0 missing)
      mxPH < 7.94    to the left,  improve=0.14900670, (0 missing)
      NO3  < 0.18    to the right, improve=0.11564070, (0 missing)
      oPO4 < 45.1    to the left,  improve=0.11106510, (0 missing)
      Chla < 12.21   to the left,  improve=0.09817759, (0 missing)
  Surrogate splits:
      PO4   < 70.465  to the left,  agree=0.844, adj=0.590, (0 split)
      oPO4  < 19.8635 to the left,  agree=0.835, adj=0.566, (0 split)
      NH4   < 46.35   to the left,  agree=0.807, adj=0.494, (0 split)
      Chla  < 2.225   to the left,  agree=0.807, adj=0.494, (0 split)
      speed splits as  RRL,         agree=0.775, adj=0.410, (0 split)

Node number 2: 83 observations
  mean=1.889157, MSE=14.38121

Node number 3: 135 observations
  mean=11.17037, MSE=175.8068
```

```
rpart.plot::prp(regression.tree.a2.final, extra=101,
                box.col="orange", split.box.col="gray")
```



The resulting tree is not nearly as complex as the original tree (hence discourages overfitting) but is still more complex than the pruned tree (which should improve predicting accuracy).

20.6.2 Model Evaluation

At this stage, we know that the linear model is not great for `a2`, and we have seen how to grow a regression tree for `a2` but we have not

yet discussed whether this model is a good fit, to say nothing of the remaining 6 algae concentrations. Can we get a better handle on these models' performance (i.e., comparing the model predictions to the real values of the target variable in the test data)?

We have discussed various metrics that can be used to determine how the values compare in Chapter 19; in this case, we elect to use the **normalized mean squared error** (NMSE):

$$\frac{\text{MSE}}{\text{mean}\left\{\left(\overline{\text{real}} - \text{real}_i\right)^2 ; i = 1, ..., N\right\}}.$$

As the ratio of MSE to a baseline predictor (the mean of the value of the target), NMSE is **unitless**. NMSE values between 0 and 1 (smaller is better) indicate that the model performs better than the baseline; greater than 1 indicate that the model's performance is sub-par.

We use the `performanceEstimation` library to run $5 \times 10-$fold cross-validations to determine which of the models (linear model and 4 regression trees parametrized by `se`) yields an optimal (smaller) NMSE value when trying to predict `a2`.

```
library(performanceEstimation)
kCV.results.algae.a2 <- performanceEstimation(
  PredTask(a2 ~ season + size + speed + mxPH + mnO2 + Cl +
    NO3 + NH4 + oPO4 + PO4 + Chla, data=algae.train, "a2"),
  c(Workflow(learner="lm",post="onlyPos"),
    workflowVariants(learner="rpartXse",
       learner.pars=list( se=c(0,0.25,0.5,0.75,1) ))),
  EstimationTask(metrics="nmse",
                 method=CV(nReps=5,nFolds=10))
)
```

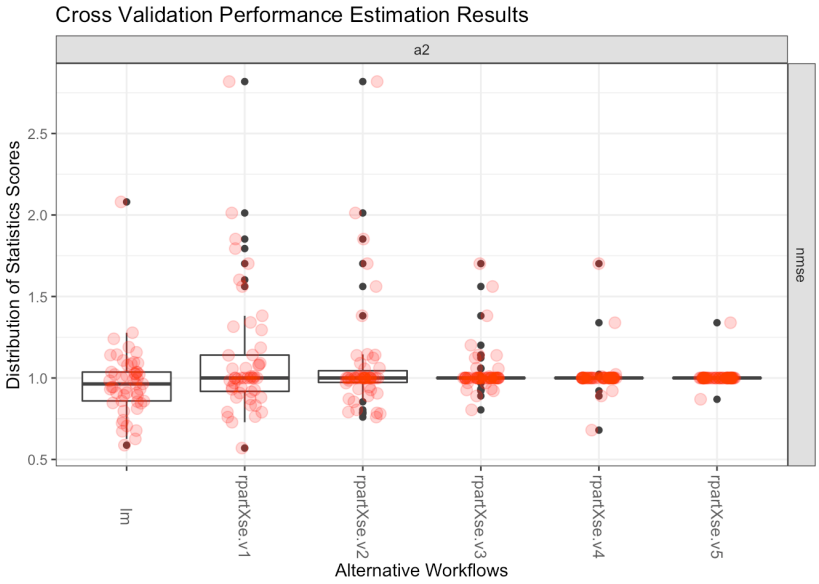
A summary and plot of the cross-validation results for NMSE can be displayed using calls to `summary()` and `plot()`.

```
summary(kCV.results.algae.a2)
```

```
== Summary of a  Cross Validation Performance Estimation Experiment ==
Task for estimating nmse using 5 x 10-Fold Cross Validation (seed=1234)

* Predictive Tasks ::  a2
* Workflows  ::  lm, rpartXse.v1, rpartXse.v2, rpartXse.v3,
                 rpartXse.v4, rpartXse.v5

-> Task:  a2
  *Workflow: lm
             nmse
avg     0.9723125
std     0.2221976
med     0.9634147
iqr     0.1771688
```

```
min     0.5878283
max     2.0801221
invalid 0.0000000

  *Workflow: rpartXse.v1
            nmse
avg     1.1148436
std     0.3871551
med     1.0000000
iqr     0.2226673
min     0.5701226
max     2.8186400
invalid 0.0000000

  *Workflow: rpartXse.v2
             nmse
avg     1.08587675
std     0.35111303
med     1.00000000
iqr     0.07178237
min     0.76004730
max     2.81864005
invalid 0.00000000

  *Workflow: rpartXse.v3
               nmse
avg     1.035773e+00
std     1.470430e-01
med     1.000000e+00
iqr     2.220446e-16
min     8.044770e-01
max     1.701835e+00
invalid 0.000000e+00

  *Workflow: rpartXse.v4
               nmse
avg     1.011250e+00
std     1.214329e-01
med     1.000000e+00
iqr     2.220446e-16
min     6.800497e-01
max     1.701835e+00
invalid 0.000000e+00

  *Workflow: rpartXse.v5
               nmse
avg     1.004167e+00
std     5.174279e-02
med     1.000000e+00
iqr     2.220446e-16
min     8.692699e-01
max     1.339067e+00
invalid 0.000000e+00
```

```
plot(kCV.results.algae.a2)
```



It is not obvious which of the models has smaller values of NMSE, although it does seem that the latter versions of the regression tree models are not substantially better than the baseline model.

The first regression tree model sometimes produces very small NMSE values, but that is offset by some of the larger values it also produces.[81]

81: Similarly for the linear model.

At any rate, visual evidence seems to suggest that the **linear model** is the best predictive model for `a2` given the training data (in this version of *k*CV), which is corrobated by a call to `topPerformers()`.

```
topPerformers(kCV.results.algae.a2)
```

```
$a2
     Workflow Estimate
nmse       lm    0.972
```

This might seem disheartening at first given how poorly the linear model performed, but it might be helpful to remember that there is no guarantee that a decent predictive model even exists in the first place.

Furthermore, regression trees and linear models are only two of a whole collection of possible models. How do **support vector machines** perform the task, for instance?[82]

82: See Chapter 21 for an in-depth discussion on the topic.

This time, however, we will learn models and perform evaluation for all target variables (`a1-a7`) simultaneously. This does not mean that we are looking for a single model which will optimize all learning tasks at once, but rather that we can prepare and evaluate the models for each target variable with the same bit of code.

This first require some code to create the appropriate model formulas (`a1 ~ .` , ... ,`a7 ~ .`) and the appropriate training data.

```
gg <- function(x,list.of.variables){
        PredTask(as.formula(paste(x,"~ .")), algae.train[,c(list.of.variables,x )],
                 x, copy=TRUE)}
(data.sources <- sapply(names(algae.train[12:18]), gg, names(algae.train[1:11])))
```

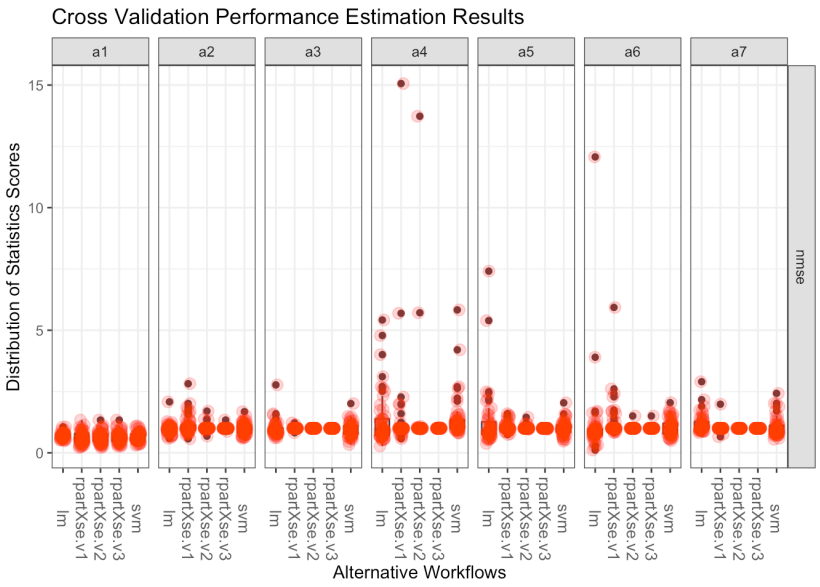
```
$a1
Prediction Task Object:
    Task Name         :: a1
    Task Type         :: regression
    Target Feature    :: a1
    Formula           :: a1 ~ .

$a2
Prediction Task Object:
    Task Name         :: a2
    Task Type         :: regression
    Target Feature    :: a2
    Formula           :: a2 ~ .

$a3
Prediction Task Object:
    Task Name         :: a3
    Task Type         :: regression
    Target Feature    :: a3
    Formula           :: a3 ~ .

$a4
Prediction Task Object:
    Task Name         :: a4
    Task Type         :: regression
    Target Feature    :: a4
    Formula           :: a4 ~ .

$a5
Prediction Task Object:
    Task Name         :: a5
    Task Type         :: regression
    Target Feature    :: a5
    Formula           :: a5 ~ .

$a6
Prediction Task Object:
    Task Name         :: a6
    Task Type         :: regression
    Target Feature    :: a6
    Formula           :: a6 ~ .

$a7
Prediction Task Object:
    Task Name         :: a7
    Task Type         :: regression
    Target Feature    :: a7
    Formula           :: a7 ~ .
```

We shall use `e1071`'s implementation of `svm()`, with various values of the `svm()`-specific parameters `cost` and `gamma`.

```
library(e1071)
kCV.results.algae.all <- performanceEstimation(
    data.sources,
    c(Workflow(learner="lm", post="onlyPos"),
      Workflow(learner="svm", learner.pars=list(
          cost=c(10,1,0.1), gamma=0.1)),
    workflowVariants(learner="rpartXse", learner.pars=list(
          se=c(0,0.7,1)))),
    EstimationTask(metrics="nmse",
                   method=CV(nReps=5, nFolds=10)))
```

The rest of the evaluation proceeds much as before, except that we can display results for the 7 target variables simultaneously.

```
plot(kCV.results.algae.all)
```




```
rankWorkflows(kCV.results.algae.all,top=3)
```

```
$a1$nmse
     Workflow  Estimate
1 rpartXse.v1 0.6163406
2 rpartXse.v2 0.6278027
3 rpartXse.v3 0.6430736

$a2$nmse
     Workflow  Estimate
1          lm 0.9723125
2         svm 0.9954432
3 rpartXse.v3 1.0041667
```

```
$a3$nmse
     Workflow  Estimate
1         svm 0.9497730
2          lm 0.9801662
3 rpartXse.v2 1.0000000

$a4$nmse
     Workflow Estimate
1 rpartXse.v3 1.001453
2 rpartXse.v2 1.351494
3          lm 1.357243

$a5$nmse
     Workflow  Estimate
1         svm 0.9968475
2 rpartXse.v3 0.9990465
3 rpartXse.v2 1.0194733

$a6$nmse
     Workflow Estimate
1 rpartXse.v2 1.010069
2 rpartXse.v3 1.010069
3         svm 1.054975

$a7$nmse
     Workflow Estimate
1 rpartXse.v2  1.00000
2 rpartXse.v3  1.00000
3 rpartXse.v1  1.00797
```

```
topPerformers(kCV.results.algae.all)
```

```
$a1
        Workflow Estimate
nmse rpartXse.v1    0.616

$a2
     Workflow Estimate
nmse       lm    0.972

$a3
     Workflow Estimate
nmse      svm     0.95

$a4
        Workflow Estimate
nmse rpartXse.v3    1.001

$a5
     Workflow Estimate
nmse      svm    0.997
```

```
$a6
        Workflow Estimate
nmse rpartXse.v2     1.01

$a7
        Workflow Estimate
nmse rpartXse.v2        1
```

For `a1`, the models seem to perform reasonably well, but it is not as rosy for the other target variables, where the baseline model is sometimes better.[83]

83: Again, this could be built-in in the data, but we might benefit from incorporating more models.

```
library(randomForest)
kCV.algae.all.rf <- performanceEstimation(
    data.sources,
    c(Workflow(learner="lm", post="onlyPos"),
      Workflow(learner="svm", learner.pars=list(
            cost=c(10,1,0.1), gamma=0.1)),
      workflowVariants(learner="rpartXse",
            learner.pars=list(se=c(0,0.7,1))),
      workflowVariants(learner="randomForest",
            learner.pars=list(ntree=c(200,500,700)))),
    EstimationTask(metrics="nmse", method=CV(nReps=5,
                                             nFolds=10))
)
```

```
rankWorkflows(kCV.algae.all.rf,top=3)
```

```
$a1$nmse
         Workflow  Estimate
1 randomForest.v2 0.5217204
2 randomForest.v3 0.5228744
3 randomForest.v1 0.5264328

$a2$nmse
         Workflow  Estimate
1 randomForest.v3 0.7798749
2 randomForest.v2 0.7806831
3 randomForest.v1 0.7849360

$a3$nmse
         Workflow  Estimate
1 randomForest.v3 0.9377108
2 randomForest.v2 0.9400108
3 randomForest.v1 0.9431801

$a4$nmse
         Workflow Estimate
1     rpartXse.v3 1.001453
2 randomForest.v3 1.006496
3 randomForest.v1 1.006806
```

```
$a5$nmse
         Workflow  Estimate
1 randomForest.v1 0.7626241
2 randomForest.v2 0.7675794
3 randomForest.v3 0.7681834

$a6$nmse
         Workflow  Estimate
1 randomForest.v2 0.8590227
2 randomForest.v3 0.8621478
3 randomForest.v1 0.8663869

$a7$nmse
     Workflow Estimate
1 rpartXse.v2  1.00000
2 rpartXse.v3  1.00000
3 rpartXse.v1  1.00797
```

`rankWorkflows()` does not report on the standard error, so we cannot tell whether the differences between the score of the best model and the other models is statistically significant.

`randomForest.v3` seems to have the best ranking across all learning tasks, so we will use it as the baseline model.

```
p <- pairedComparisons(kCV.algae.all.rf,
                       baseline="randomForest.v3")
p$nmse$F.test
p$nmse$BonferroniDunn.test
```

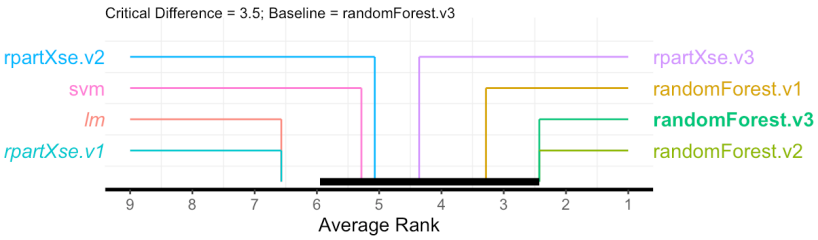
```
$chi
[1] 22.86905

$FF
[1] 5.251025

$critVal
[1] 0.7071231

$rejNull
[1] TRUE

$critDif
[1] 3.52218

$baseline
[1] "randomForest.v3"

$rkDifs
             lm             svm     rpartXse.v1     rpartXse.v2     rpartXse.v3
      4.1428571       2.8571429       4.1428571       2.6428571       1.9285714
randomForest.v1 randomForest.v2
      0.8571429       0.0000000
```

```
$signifDifs
            lm             svm     rpartXse.v1     rpartXse.v2     rpartXse.v3
          TRUE           FALSE            TRUE           FALSE           FALSE
randomForest.v1 randomForest.v2
         FALSE           FALSE
```

We can reject with 95% certainty the hypothesis that the performance of the baseline method (`randomForest.v3`) is the same as that of the linear model and the first 2 regression trees, but not that it is better than `svm`, `rpartXse.v3`, and the other 2 random forests.

The information is also displayed in the Bonferroni-Dunn CD diagram below.

```
CDdiagram.BD(p)
```



20.6.3 Model Predictions

Finally, we might actually be interested in generating predictions for each of the target variables in the testing set. This simply requires that the best performers for each target response be brought together in an `R` object.

```
best.performers <- sapply(taskNames(kCV.algae.all.rf),
        function(x) topPerformer(kCV.algae.all.rf,
                metric="nmse", task=x)
best.performers
```

```
$a1
Workflow Object:
    Workflow ID       ::  randomForest.v2
    Workflow Function ::  standardWF
         Parameter values:
         learner.pars  -> ntree=500
         learner  -> randomForest

$a2
Workflow Object:
    Workflow ID       ::  randomForest.v3
    Workflow Function ::  standardWF
         Parameter values:
         learner.pars  -> ntree=700
         learner  -> randomForest
```

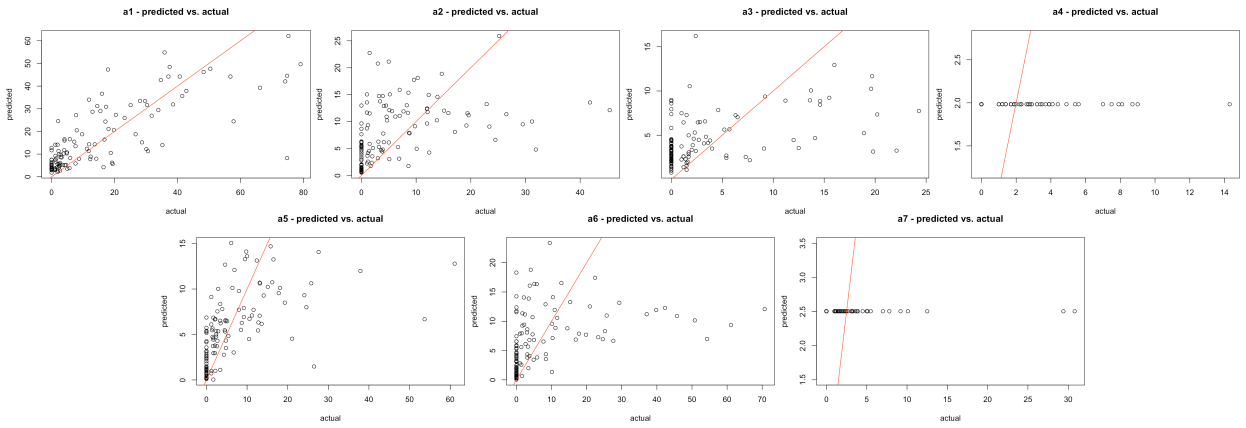
```
$a3
Workflow Object:
    Workflow ID       ::  randomForest.v3
    Workflow Function ::  standardWF
         Parameter values:
         learner.pars  -> ntree=700
         learner  -> randomForest

$a4
Workflow Object:
    Workflow ID       ::  rpartXse.v3
    Workflow Function ::  standardWF
         Parameter values:
         learner.pars  -> se=1
         learner  -> rpartXse

$a5
Workflow Object:
    Workflow ID       ::  randomForest.v1
    Workflow Function ::  standardWF
         Parameter values:
         learner.pars  -> ntree=200
         learner  -> randomForest

$a6
Workflow Object:
    Workflow ID       ::  randomForest.v2
    Workflow Function ::  standardWF
         Parameter values:
         learner.pars  -> ntree=500
         learner  -> randomForest

$a7
Workflow Object:
    Workflow ID       ::  rpartXse.v2
    Workflow Function ::  standardWF
         Parameter values:
         learner.pars  -> se=0.7
         learner  -> rpartXse
```

The observations that form the **testing set** are placed in an object, as below:

```
test.observations <- array(dim=c(nrow(algae.test),7,2),
    dimnames=list(rownames(algae.test), paste("a",1:7),
        c("actual","predicted")))
```

The function `runWorkflow()` will compute the predicted values for each of the targets' best performers. We can then plot the predicted and actual values for each of the testing set targets.

```
for(j in 1:7){
    results <- runWorkflow(best.performers[[j]],
        as.formula(paste(names(best.performers)[j],"~ .")),
        algae.train[,c(1:11,11+j)],
        algae.test[,c(1:11,11+j)])
    test.observations[,j,"actual"] <- results$trues
    test.observations[,j,"predicted"] <- results$preds
}

df.a1 <- as.data.frame(test.observations[,1,])
df.a2 <- as.data.frame(test.observations[,2,])
df.a3 <- as.data.frame(test.observations[,3,])
df.a4 <- as.data.frame(test.observations[,4,])
df.a5 <- as.data.frame(test.observations[,5,])
df.a6 <- as.data.frame(test.observations[,6,])
df.a7 <- as.data.frame(test.observations[,7,])

plot(df.a1,main="a1 - predicted vs. actual")
abline(0,1,col="red")
plot(df.a2,main="a2 - predicted vs. actual")
abline(0,1,col="red")
plot(df.a3,main="a3 - predicted vs. actual")
abline(0,1,col="red")
plot(df.a4,main="a4 - predicted vs. actual")
abline(0,1,col="red")
plot(df.a5,main="a5 - predicted vs. actual")
abline(0,1,col="red")
plot(df.a6,main="a6 - predicted vs. actual")
abline(0,1,col="red")
plot(df.a7,main="a7 - predicted vs. actual")
abline(0,1,col="red")
```



The models simply are not that great, but we already expected that. The average prediction for each target is shown below.

```
(average.prediction <- apply(algae.train[,12:18],2, mean))
```

```
    a1    a2    a3    a4    a5    a6    a7
17.47  7.64  4.13  1.98  4.96  5.81  2.50
```

Finally, you might be interested in the NMSE metrics for the predicted values and how they compare to the NMSE metrics on the training set. Is any of this surprising?

```
apply((test.observations[,,"actual"] - test.observations[,,"predicted"])^2, 2, sum) /
    apply((scale(test.observations[,,"actual"], average.prediction,FALSE))^2, 2, sum)
```

```
  a1   a2   a3   a4   a5   a6   a7
0.40 0.88 0.78 1.00 0.71 0.84 1.00
```

20.7 Exercises

1. Let (X, Y) be a bivariate normal random variable with parameters

$$\mu_X = 12, \mu_Y = -7, \sigma_X^2 = 1, \sigma_Y^2 = 2, \sigma_{XY} = 4.$$

   Consider the parameter

$$\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}.$$

   Using a bootstrap procedure with $N = 100$ samples and $M = 200$ replicates, provide a confidence interval for the true value of α. [5]
2. Explicitly obtain the polynomial regression models in the Gapminder Example, for $d = 2, 3, 4$.
3. Play around with a variety of knots in the step function regression models for the Gapminder Example, and build the corresponding confidence intervals (including those of the example). How would you determine the number and location of the knots?
4. Determine the optimal number of knots K for cubic splines and natural cubic splines for the Gapminder Example, using cross-validation.
5. Build piecewise cubic splines and continuous piecewise cubic splines for the Gapminder Example. Use cross-validation to determine the optimal number of knots.
6. Predict life expectancy of countries in 2011 using the various spline models (in the text and in the exercises) on the Gapminder dataset, with training/testing pairs. Evaluate your models. Which ones perform best?
7. Predict life expectancy of countries in 2011 using various GAM models on the Gapminder dataset, with training/testing pairs. Evaluate your models. Which ones perform best?
8. Consider the dataset `algae_blooms.csv`, as in Section 20.6. Run the analysis with a scaled dataset. Run the analysis with a PCA-reduced dataset. Do the results change significantly?
9. Consider the following datasets:
   - `GlobalCitiesPBI.csv`
   - `2016collisionsfinal.csv`
   - `polls_us_election_2016.csv`
   - `HR_2016_Census_simple.xlsx`, and
   - `UniversalBank.csv`.

   For each of these datasets, identify a response variable (or more than one, if the fancy strikes you) and predictors, and build models to predict the response(s) using the various methods discussed in this chapter. Evaluate and rank the resulting models. You may need to clean, transform, and visualize the data along the way.
10. Complete the definition of the `Python` function `kfoldCV(k, data, yname, formulas)` where `k` is the number of folds, `data` is the data set, `yname` is the column name of the dependent variable, and `formulas` is a list of formulas. The function should return the tuple `fit, f` where `fit` is the OLS model for the formula `f` in `formulas` that has the minimum $k-$fold CV estimate. Use it on the `mpg` data set with $k = 10$ to obtain a good model for predicting `mpg`.

```
import seaborn as sns

df = sns.load_dataset('mpg')
df.head()

def kfoldCV(k, data, yname, formulas):
    fit = None
    # Your code here.  Don't forget to obtain a
    # random permutation of the observations

    for f in formulas:
        # Your code here
        None

    return fit, f
```

Chapter References

[1] B. Boehmke and B. Greenwell. *Hands on Machine Learning with R* . CRC Press.

[2] G.E.P. Box. 'Use and Abuse of Regression'. In: *Journal of Technometrics* 8.4 (Nov. 1966), pp. 625–629.

[3] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction , 2nd ed.* Springer, 2008.

[4] T. Hastie, R. Tibshirani, and M. Wainwright. *Statistical Learning with Sparsity : the LASSO and Generalizations*. Monographs on statistics and applied probability, no. 143. CRC Press, 2015.

[5] G. James et al. *An Introduction to Statistical Learning: With Applications in R* . Springer, 2014.

[6] M.H. Kutner et al. *Applied Linear Statistical Models*. McGraw Hill Irwin, 2004.

[7] D. Robinson. 'What's the difference between data science, machine learning, and artificial intelligence? '. In: *Variance Explained* (Jan. 2018).

[8] H. Rosling. *The Health and Wealth of Nations* . Gapminder Foundation, 2012.

[9] H. Rosling, O. Rosling, and A.R. Rönnlund. *Factfulness: Ten Reasons We're Wrong About The World - And Why Things Are Better Than You Think* . Hodder & Stoughton, 2018.

[10] H. Sahai and M.I. Ageel. *The Analysis of Variance: Fixed, Random and Mixed Models*. Birkhäuser, 2000.

[11] L. Torgo. *Data Mining with R, 2nd ed.* CRC Press, 2016.

[12] D.H. Wolpert and W.G. Macready. 'Coevolutionary free lunches'. In: *IEEE Transactions on Evolutionary Computation* 9.6 (2005), pp. 721–735. DOI: 10.1109/TEVC.2005.856205.

[13] D.H. Wolpert and W.G. Macready. 'No free lunch theorems for optimization'. In: *IEEE Transactions on Evolutionary Computation* (1997).

Focus on Classification and Supervised Learning

21

by **Patrick Boily**, with contributions from **Olivier Leduc** and **Shintaro Hagiwara**

In Chapter 19 (*Machine Learning 101*), we provided a (mostly) math-free general overview of machine learning. In this chapter, we present an introductory mathematical treatment of the discipline, with a focus on classification, ensemble learning, and non-parametric supervised methods.

Our approach once again borrows heavily from [14, 18]; explanations and examples are also available in [3, 6]. It provides a continuation of the treatment found in Chapter 20 (*Regression and Value Estimation*) and is a companion piece to Chapter 22 (*Focus on Clustering*).

21.1 Overview

We will discuss classification in the same context as we discussed regression/value estimation in Section 20.1; the latter should have been read before embarking on this chapter.[1] In particular, it is expected that readers are familiar with training sets Tr and testing sets Te for a dataset with n observations $\mathbf{x}_1, \ldots, \mathbf{x}_n$, p predictors $X_1, \ldots, X_p$ and response variable Y (see Figure 20.4 for details).

1: Or, at the very least, should be read concurrently.

One important note about this chapter's notation: in the design matrix $\mathbf{X}$, a row corresponds to the signature vector of an observation (the values of the predictors); when we write $\mathbf{x}$ or $\boldsymbol{\beta}$, we typically understand those to be **column vectors**. When in doubt, remember that all matrices/vectors involved must have compatible dimensions when multiplied or compared; that will sometimes mean that vectors must be viewed as row-vectors rather than column vectors, and *vice-versa*, depending on the context.

21.1.1 Formalism

In a **classification setting**, the response Y is **categorical**, which is to say that $Y \in \mathcal{C}$, where $\mathcal{C} = \{C_1, \ldots, C_K\}$, but the supervised learning objectives remain the same:

- build a classifier $C(\mathbf{x}^*)$ that assigns a label $C_k \in \mathcal{C}$ to test observations $\mathbf{x}^*$;
- understand the role of the predictors in this assignment, and
- assess the uncertainty and the accuracy of the classifier.

The main difference with the regression setting (and to be fair, it's a big one) is that we do not have access to an MSE-type metric to evaluate the classifier's performance.

The counterpart of the regression function

$$f(\mathbf{x}) = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}]$$

is defined as follows. For $1 \leq k \leq K$, let $p_k(\mathbf{x}) = P(Y = C_k \mid \vec{X} = \mathbf{x})$;[2] the **Bayes optimal classifier** at $\mathbf{x}$ is the function

$$C(\mathbf{x}) = C_j \quad \text{where } p_j(\mathbf{x}) = \max\{p_1(\mathbf{x}), \ldots, p_K(\mathbf{x})\}.$$

As was the case or regression, it could be that there are too few observations at $\vec{X} = \mathbf{x}$ to estimate the probability exactly, in which case we might want to allow for **nearest neighbour averaging**:

$$\hat{C}(\mathbf{x}) = C_j, \quad \text{where } \tilde{p}_j(\mathbf{x}) = \max\{\tilde{p}_1(\mathbf{x}), \ldots, \tilde{p}_K(\mathbf{x})\},$$

and $\tilde{p}_k(\mathbf{x}) = P(Y = C_k \mid \vec{X} \in N(\mathbf{x}))$ and $N(\mathbf{x})$ is a neighbourhood of $\mathbf{x}$.[3]

The quantity that plays an analogous role to the MSE for $\tilde{C}(\mathbf{x})$ is the **misclassification error rate**:

$$\mathrm{ERR}_{\mathrm{Te}} = \frac{1}{M} \sum_{j=N+1}^{N+M} \mathcal{I}[y_j \neq \tilde{C}(\mathbf{x}_j)],$$

where $\mathcal{I}$ is the **indicator function**

$$\mathcal{I}[\text{condition}] = \begin{cases} 0 & \text{if the condition is false} \\ 1 & \text{otherwise} \end{cases}$$

The Bayes optimal classifier $C(\mathbf{x})$ is the optimal classifier with respect to $\mathrm{ERR}_{\mathrm{Te}}$; the **Bayes error rate**

$$\eta_{\mathbf{x}} = 1 - \mathrm{E}\left[\max_k P(Y = C_k \mid \vec{X} = \mathbf{x})\right]$$

corresponds to the irreducible error and provides a lower limit on any classifier's expected error.

Most classifiers build structured models $\hat{C}(\mathbf{x})$ which **directly** approximate the Bayes optimal classifier $C(\mathbf{x})$ (such as support vector machines or naïve Bayes classifiers), but some classifiers build structured models $\hat{p}_k(\mathbf{x})$ for the **conditional probabilities** $p_k(\mathbf{x})$, $1 \leq j \leq K$, which are then used to build $\hat{C}(\mathbf{x})$, such as logistic regression, generalized additive models, and $k-$nearest neighbours.

The latter models are said to be **calibrated** (i.e., the relative values of $\hat{p}_k(\mathbf{x})$ represent relative probabilities), whereas the former are **non-calibrated**.[4]

2: In other words, pick the most numerous categorical label of observations for which the signature vector is $\mathbf{x}$.

3: The curse of dimensionality is also in play when p becomes too large.

4: Only the most likely outcome is provided; it is impossible to say to what extent a given outcome is more likely than another.

21.1.2 Model Evaluation

The **confusion matrix** of a classifier on Te is a tool to evaluate the model's performance:

		prediction	
		0	1
actual	0	TP	FN
	1	FP	TN

Here, TP stands for **true positive**, FN for **true negative**, FP for **false positive**, and TN for **true negative**. There are various **classifier evaluation metrics**; if the testing set Te has M observations, then:

- **accuracy** measures the correct classification rate $\frac{\text{TP+TN}}{M}$;
- **misclassification** is $\frac{\text{FP+FN}}{M} = 1 - \text{accuracy}$;
- **false positive rate** (FPR) is $\frac{\text{FP}}{\text{FP+TN}}$;
- **false negative rate** (FNR) is $\frac{\text{FN}}{\text{TP+FN}}$;
- **true positive rate** (TPR) is $\frac{\text{TP}}{\text{TP+FN}}$;
- **true negative rate** (TNR) is $\frac{\text{TN}}{\text{FP+TN}}$;

There are other measures, including the F_1-score, the Matthews' correlation coefficient, etc. [28].

One thing to remember is that we should not put all the performance evaluation eggs in the same metric basket!

21.1.3 Bias-Variance Trade-Off

The **bias-variance trade-off** (see Section 20.1) is also observed in classifiers, although the decomposition is necessarily different (see [14] for details).

In a $k-$**nearest neighbours** classifier, for instance, the prediction for a new observation with predictors $\mathbf{x}^* \in$ Te is obtained by finding the most frequent class label of the k nearest neighbours to $\mathbf{x}^*$ in Tr on which the model $\hat{C}_{k\text{NN}}(\mathbf{x})$ is built.

As the number of nearest neighbours under consideration increases, the complexity of the model $\hat{C}_{k\text{NN}}(\mathbf{x})$ decreases, and *vice-versa*.

We would thus expect:

- a model with a large k to underfit the data;
- a model with a small k to overfit the data, and
- models in the "Goldilock zone" to strike a balance between **prediction accuracy** and **interpretability of the decision boundary** (see Figures 20.5 and 21.1).

As it happens, the optimal classifier $Y = C(\vec{X})$ is, in fact, the Bayes optimal classifier.

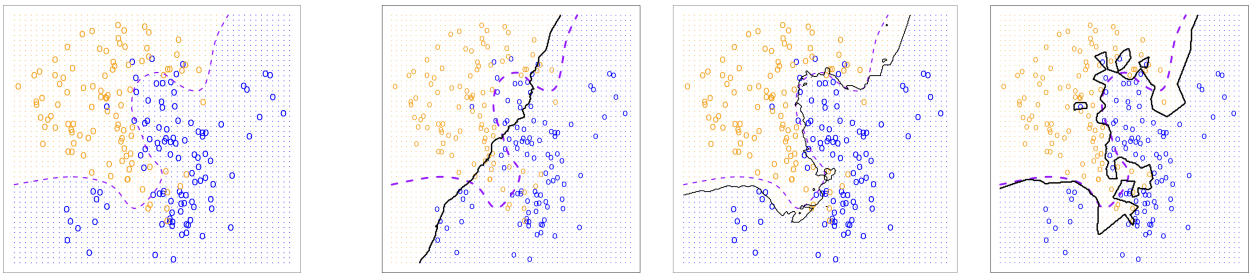


Figure 21.1: Illustration of the accuracy-boundary interpretability trade-off for classifiers on an artificial dataset; Bayes optimal classifier $C(\mathbf{x})$ (leftmost), underfit $\hat{C}_{100\text{NN}}(\mathbf{x})$ model (2nd leftmost), Goldilock $\hat{C}_{10\text{NN}}(\mathbf{x})$ model (3rd leftmost), overfit $\hat{C}_{1\text{NN}}(\mathbf{x})$ model (4th leftmost). Notice the interplay between prediction accuracy and complexity of the decision boundary (the dashed curve in the last three graphs shows the Bayes optimal boundary). [18, 14]

Comparison Between kNN and OLS We are going to try to get a better intuitive sense of the bias-variance trade-off by comparing ordinary least squares (OLS), a rigid yet simple model (as measured by the number of **effective parameters**), with $k-$nearest neighbours (kNN), a very flexible yet more complex model (again, according to the number of effective parameters).

Given an input vector $\mathbf{z} \in \mathbb{R}^p$, the $k-$**nearest neighbours** (kNN) model predicts the response Y as the average

$$\hat{Y} = \text{Avg}\{Y(\mathbf{x}) \mid \mathbf{x} \in N_k(\mathbf{z})\} = \frac{1}{k} \sum_{\mathbf{x} \in N(\mathbf{z})} Y(\mathbf{x}),$$

where $Y(\mathbf{x})$ is the known response for predictor $\mathbf{x} \in \text{Tr}$ and $N_k(\mathbf{z})$ is the set of the k training observations **nearest to z**. Another approach to neighbourhoods, which we will use at a later stage, is that they contain all training observations **within a certain (fixed) distance** of $\mathbf{z}$.[5]

5: The notion of proximity depends on the distance metric in use; the Euclidean case is the most common, but it does not have to be that one.

For classification problems, kNN models use the mode instead of the average. Of course, the prediction may depend on the value of k: in the **classification** image below, the 6NN prediction would be a red star, whereas the 19NN model prediction would be a blue disk.

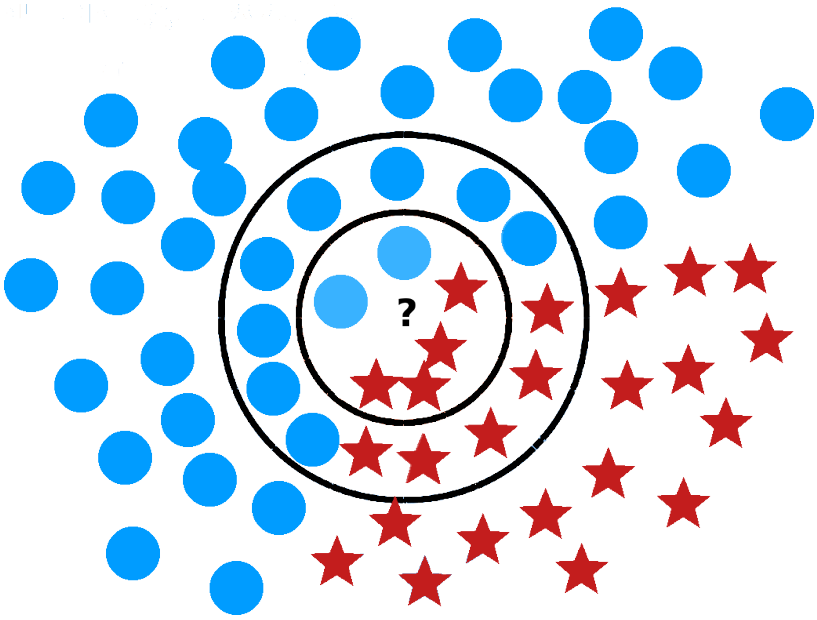


Figure 21.2: Illustration of kNN classifiers.

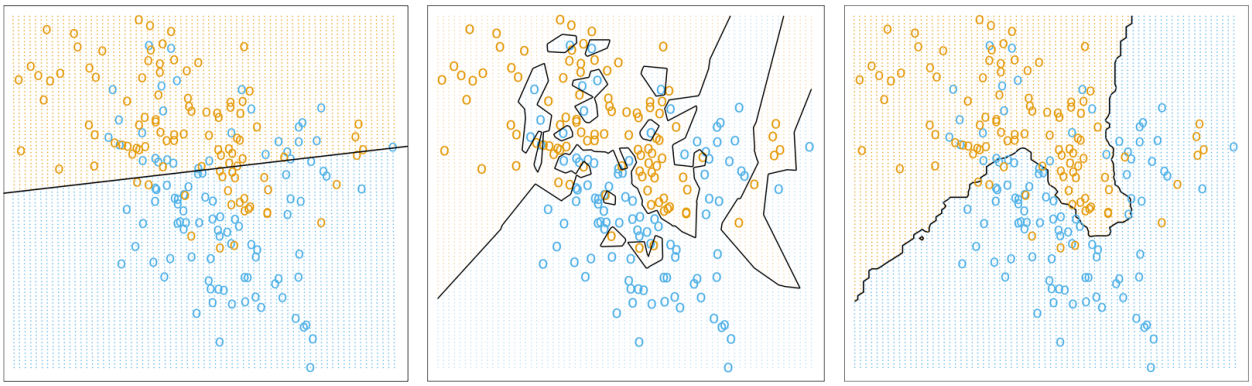


Figure 21.3: Classification based on OLS (left), 1NN (middle), and 15NN (right). [18, 14]

The following classification example (based on [14]) illustrates some of the bias-variance trade-off consequences. Consider a training dataset Tr consisting of 200 observations with features $(x_1, x_2) \in \mathbb{R}^2$ and responses $y \in \{\text{BLUE}_{(=0)}, \text{ORANGE}_{(=1)}\}$. Let $[\cdot] : \mathbb{R} \to \{\text{BLUE}, \text{ORANGE}\}$ denote the function

$$[w] = \begin{cases} \text{BLUE} & w \leq 0.5 \\ \text{ORANGE} & w > 0.5 \end{cases}$$

Linear Fit Fit an OLS model

$$\hat{y}(\mathbf{x}) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2$$

on Tr; the class prediction is $\hat{g}(\mathbf{x}) = [\hat{y}(\mathbf{x})]$. The **decision boundary**

$$\partial_{\text{OLS}} = \{(x_1, x_2) \mid \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = 0.5\}$$

is shown in Figure 21.3 (on the left); it is a straight line which can be described using only 2 **effective parameters**.[6]

There are several **misclassifications** on both sides of ∂_{OLS}; even though errors seem to be unavoidable, the OLS model is likely to be **too rigid**.

6: Their number is a measure of a model's **complexity**.

kNN Fit If $\hat{y}(\mathbf{x})$ represents the proportion of ORANGE points in $N_k(\mathbf{x})$, then the class prediction is $\hat{g}(\mathbf{x}) = [\hat{y}(\mathbf{x})]$. The decision boundaries $\partial_{1\text{NN}}$ and $\partial_{15\text{NN}}$ are displayed in Figure 21.3.

They are both irregular: $\partial_{1\text{NN}}$ is overfit, whereas $\partial_{15\text{NN}}$ is less likely to be.[7] The effective parameters are not as obviously defined for this model; one approach is to view kNN as a model that fits 1 parameter (a mean) to each **ideal** (non-overlapping) **neighbourhood** in the data, so that the number of effective parameters is roughly equal to the number of such neighbourhoods:

$$\frac{N}{k} \approx \begin{cases} 13 & \text{when } k = 15 \\ 200 & \text{when } k = 1 \end{cases}$$

The kNN models are thus fairly complex, in comparison with the OLS model. There are no misclassification for $k = 1$, and several in the case $k = 15$.[8] The 15NN model seems to strike a balance between various competing properties; it is likely nearer the "sweet spot" of the test error curve.[9]

7: Although neither is great for **interpretability**.

8: But not as many as with the OLS model.

9: Remember however that we have not evaluated the performance of the models on a testing set Te; we have only described some of their behaviours on the training set Tr.

Conclusions The OLS model is **stable** as adding a few training observations is unlikely to alter the fit substantially, but also **biased** since the assumptions of a valid linear fit is questionable; the kNN models are **unstable** as adding a few training observations is quite likely to alter the fit substantially (especially for small values of k)), but it is also **unbiased** since no apparent assumptions are made about the data.

So which approach is best? That depends entirely on what the ultimate task is: description, prediction, etc. In predictive data science, machine learning, and artificial intelligence, the validity of modeling assumptions takes a backseat to a model's ability to **make good predictions on new (and unseen) observations**.

Naturally, we would expect that models whose assumptions are met are more likely to make good predictions than models for whom that is not the case, but it does not need to be the case. The theory of linear models is mature and extensive, and we could have discussed a number of their other features and extensions (see Chapter 8 for details).

Keep in mind, then, that machine learning methods are not meant to replace or supplant classical statistical analysis methods, but rather, to **complement them**. They simply provide different approaches to **gain insights from data**.

21.2 Simple Classification Methods

Qualitative variables take values in an **unordered** set $\mathscr{C} = \{C_1, \ldots, C_K\}$. For instance,

- hair colour $\in$ {black, red, blond, grey, other}
- email message $\in$ {ham, spam}
- life expectancy $\in$ {high, low}

For a training set Tr with observations $(\vec{X}, Y) \in \mathbb{R}^p \times \mathscr{C}$, the **classification problem** is to build a classifier $\hat{C} : \mathbb{R}^p \to \mathscr{C}$ to approximate the optimal Bayes classifier $C : \mathbb{R}^p \to \mathscr{C}$ (as discussed in the previous section).

In many instances, we might be more interested in the probabilities

$$\pi_k(\mathbf{x}) = P\{\hat{C}(\mathbf{x}) = C_k\}, \quad k = 1, \ldots, K$$

than in the classification predictions themselves. Typically, the classifier $\hat{C}$ is **built** on a training set

$$\text{Tr} = \{(\mathbf{x}_j, y_j)\}_{j=1}^{N}$$

and **evaluated** on a testing set

$$\text{Te} = \{(\mathbf{x}_i, y_i)\}_{i=N+1}^{N+M}.$$

Example Let us revisit the `gapminder.csv` dataset, again focusing on observations from 2011, with the difference that life expectancy is now recorded as "high" (1) if it falls above 72.45 (the median in 2011), and as "low" (0) otherwise.

Setting up the Gapminder dataset

```
library(dplyr)
gapminder.ML = read.csv("gapminder.csv",
                        stringsAsFactors=TRUE)
gapminder.ML <- gapminder.ML[complete.cases(gapminder.ML),]
gapminder.ML <- gapminder.ML[,c("country","year","region",
            "continent","population","infant_mortality",
            "fertility","gdp","life_expectancy")]

gapminder.2011 <- gapminder.ML |> filter(year==2011) |>
  mutate(LE=as.factor(ifelse(life_expectancy <
         median(life_expectancy),"low","high")))
```

The structure and summary are provided below:

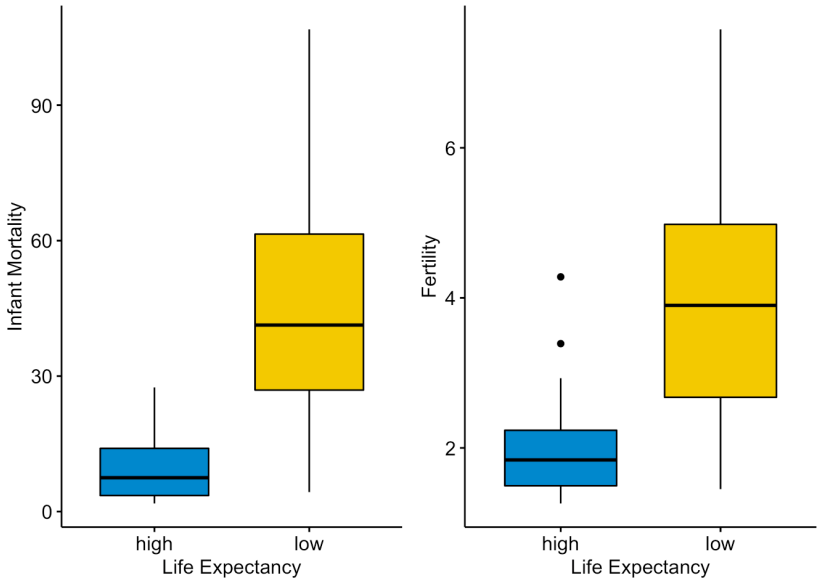
```
summary(gapminder.2011[,c("infant_mortality",
                      "fertility", "LE")])
```

```
infant_mortality    fertility        LE
Min.   :  1.800   Min.   :1.260   high:83
1st Qu.:  7.275   1st Qu.:1.792   low :83
Median : 16.900   Median :2.420
Mean   : 27.333   Mean   :2.931
3rd Qu.: 41.125   3rd Qu.:3.908
Max.   :106.800   Max.   :7.580
```

Let us assume that we are interested in modeling the response LE (Y) as a linear response of the predictors X_1 (infant mortality) and X_2 (fertility), using ordinary linear regression (OLS).[10]

10: See Chapter 8 for a detailed discussion of such models.

```
p1 <- ggpubr::ggboxplot(gapminder.2011, x = "LE",
    y = "infant_mortality", fill = "LE", palette = "jco",
    xlab="Life Expectancy", ylab="Infant Mortality") +
    ggpubr::rremove("legend")

p2 <- ggpubr::ggboxplot(gapminder.2011, x = "LE",
    y = "fertility", fill = "LE", palette = "jco",
    xlab="Life Expectancy", ylab="Fertility") +
    ggpubr::rremove("legend")

grid::pushViewport(grid::viewport(
    layout = grid::grid.layout(nrow = 1, ncol = 2)))

# helper function to define a region on the layout
define_region <- function(row, col){
  grid::viewport(layout.pos.row = row, layout.pos.col = col)
}

print(p1, vp = define_region(row = 1, col = 1))
print(p2, vp = define_region(row = 1, col = 2))
```



We run an OLS regression of Y on $\vec{X}$ over Tr and obtain the model

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 \cdot \text{infant mortality} + \hat{\beta}_2 \cdot \text{fertility},$$

from which we would classify an observation's life expectancy as

$$\hat{C}(\vec{X}) = \begin{cases} \texttt{high} & \text{if } \hat{Y} > 0.5 \\ \texttt{low} & \text{else} \end{cases}$$

```
gapminder.2011 <- gapminder.2011 |>
    mutate(LE.resp=ifelse(LE=="high",1,0))
model.class <- lm(LE.resp ~ infant_mortality + fertility,
    data=gapminder.2011)
beta_0=as.numeric(model.class[[1]][1])
beta_1=as.numeric(model.class[[1]][2])
beta_2=as.numeric(model.class[[1]][3])
model.class[[1]]
```

```
(Intercept) infant_mortality        fertility
 1.00102979      -0.01188533      -0.06010533
```

Thus,

$$\hat{Y} = 1.001 - 0.012 \cdot \text{infant mortality} - 0.060 \cdot \text{fertility}.$$

We plot the decision boundary on the scatterplot of the domain:[11]

```
slope = -beta_1/beta_2
intercept = 0.5*(1-2*beta_0)/beta_2

ggpubr::ggscatter(gapminder.2011, x="infant_mortality",
    y="fertility", shape="LE", color="LE", palette="jco",
    size = 2, xlab="Infant Mortality", ylab = "Fertility",
```

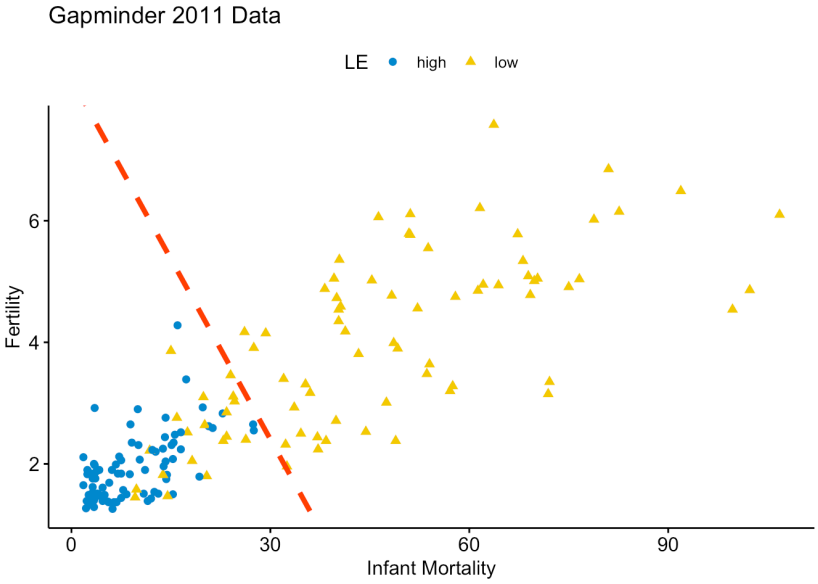
11: If the boundary splits the observations at $\hat{Y} = \gamma \in [0, 1]$, then it solves

$$\gamma = \beta_0 + \beta_1 X_1 + \beta_2 X_2, \quad \beta_2 \neq 0,$$

so that

$$X_2 = \left(\frac{\gamma - \beta_0}{\beta_2}\right) - \frac{\beta_1}{\beta_2} X_1.$$

```
    title = "Gapminder 2011 Data") +
  ggplot2::geom_abline(intercept = intercept,
    slope = slope, color="red", linetype="dashed", size=1.5)
```



The OLS approach is likely to do a decent job here since the data is roughly **linearly separable** over the predictors. This will not usually be the case, however.

In the example above, the optimal regression function is

$$f(\mathbf{x}) = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}] = P(Y = 1 \mid \vec{X} = \mathbf{x}) = p_1(\mathbf{x})$$

because Y is a binary variable; this might lead us to believe that $f(\mathbf{x})$ could also be used to directly classify and determine the class probabilities for the data, in which case there would be no need for a **separate classification apparatus**.[12]

12: There is one major drawback with this approach: if linear regression is used to model the data (which is to say, if we assume that $f(\mathbf{x}) \approx \mathbf{x}^\top \boldsymbol{\beta}$), we need to insure that $\hat{f}_{\mathrm{OLS}}(\mathbf{x}) \in [0, 1]$ for all $\mathbf{x} \in \mathrm{Te}$. This, in general, cannot be guaranteed.

A problem arises if we study the residual situation further. If we model $Y = \{0, 1\}$ with an OLS regression, we have

$$Y_i = \mathbf{x}_i^\top \boldsymbol{\beta} + \varepsilon_i.$$

Thus

$$\varepsilon_i = Y_i - \mathbf{x}_i^\top \boldsymbol{\beta} = \begin{cases} 1 - \mathbf{x}_i^\top \boldsymbol{\beta} & \text{if } Y_i = 1 \\ -\mathbf{x}_i^\top \boldsymbol{\beta} & \text{if } Y_i = 0 \end{cases}$$

But OLS assumes that $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$, which is clearly not the case here, as ε_i can only take two values. OLS is thus not an appropriate way to model the response.

Furthermore,

$$\mathrm{Var}(Y_i) = p_1(\mathbf{x}_i)(1 - p_1(\mathbf{x}_i)),$$

since Y_i is a binomial random variable, and

$$\mathrm{Var}(\varepsilon_i) = \mathrm{Var}(Y_i - p_1(\mathbf{x}_i)) = \mathrm{Var}(Y_i) = p_1(\mathbf{x}_i)(1 - p_1(\mathbf{x}_i)),$$

which is not constant as it depends on $\mathbf{x}_i$.

The OLS assumptions are thus violated at every turn[13] – OLS is simply not a good fit/modeling approach to estimate

$$p_k(\mathbf{x}) = P(Y = C_k \mid \vec{X} = \mathbf{x}).$$

We start by introducing two simple classification methods (see [18] for more details).

21.2.1 Logistic Regression

The problems presented above point to OLS not being an ideal method for classification, but the linear regression still provided a good separator in the Gapminder example. This suggests that we should not automatically reject the possibility of first transforming the data and then seeing if OLS might not provide an appropriate modeling strategy on the transformed data.

Formulations In **logistic regression**, we are seeking an invertible transformation $g : \mathbb{R} \to [0, 1]$, with $g(y^*) = y$ and $g^{-1}(y) = y^*$. The variable y must behave like a probability; in the 2-class setting, we use $g_L(y^*)$ to approximate the probability

$$p_1(\mathbf{x}) = P(Y = 1 \mid \vec{X} = \mathbf{x}).$$

The idea is to run OLS on a transformed training set

$$\text{Tr}^* = \{(\mathbf{x}_i, y_i^*)\}_{i=1}^N,$$

and to transform the results back using $y_i = g(y_i^*)$.

There are many such functions: the **probit** model,[14] which we will not discuss, and the **logit** model regression model are two common approaches.

Logit Model The logit model uses the transformation

$$y = g_L(y^*) = \frac{e^{y^*}}{1 + e^{y^*}}.$$

It is such that

$$g_L^{-1}(0) = -\infty, \quad g_L^{-1}(1) = \infty, \quad g_L^{-1}(0.5) = 0, \quad \text{etc.}$$

We solve for y^* in order to get a transformed response $y^* \in \mathbb{R}$ (instead of one restricted to $[0, 1]$):

$$p_1(\mathbf{x}) = \frac{e^{y^*}}{1 + e^{y^*}} \iff y^* = g_L^{-1}(y) = \ln\left(\frac{p_1(\mathbf{x})}{1 - p_1(\mathbf{x})}\right).$$

It is the **log-odds** transformed observations that we attempt to fit with an OLS model:

$$\hat{Y}^* = \ln\left(\frac{p_1(\mathbf{x})}{1 - p_1(\mathbf{x})}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p = \mathbf{x}^\top \boldsymbol{\beta}.$$

13: There is another way in which OLS could fail, but it has nothing to do with the OLS assumptions *per se*. When the set of qualitative responses contains more than 2 level (such as $\mathscr{C} = \{\texttt{low}, \texttt{medium}, \texttt{high}\}$, for instance), the response is usually encoded using numerals to facilitate the implementation of the analysis:

$$Y = \begin{cases} 0 & \text{if } \texttt{low} \\ 1 & \text{if } \texttt{medium} \\ 2 & \text{if } \texttt{high} \end{cases}$$

This encoding suggests an **ordering** and a **scale** between the levels (for instance, the difference between "high" and "medium" is equal to the difference between "medium" and "low", and half again as large as the difference between "high" and "low"). OLS is not appropriate in this context.

14: The probit transformation uses $g_P(y^*) = \Phi(y^*)$, where Φ is the cumulative distribution function of $\mathcal{N}(0, 1)$.

In order to make a prediction for $p_1(\mathbf{x})$, we estimate y^* and use the logit transformation to recover y. For instance, if $\mathbf{x}^\top\hat{\boldsymbol{\beta}} = 0.68$, then

$$\hat{y}^* = \ln\left(\frac{\hat{p}_1(\mathbf{x})}{1-\hat{p}_1(\mathbf{x})}\right) = 0.68$$

and

$$\hat{p}_1(\mathbf{x}) = \frac{e^{y^*}}{1+e^{y^*}} = \frac{e^{0.68}}{1+e^{0.68}} = 0.663.$$

Depending on the **decision rule threshold** γ, we may thus predict that $\hat{C}(\mathbf{x}) = C_1$ if $p_1(\mathbf{x}) > \gamma$ or $\hat{C}(\mathbf{x}) = C_2$, otherwise.

The technical challenge is in obtaining the coefficients $\hat{\boldsymbol{\beta}}$; they are found by **maximizing the likelihood** (see [17])

$$\begin{aligned} L(\boldsymbol{\beta}) &= \prod_{y_i=1} p_1(\mathbf{x}_i) \prod_{y_i=0} (1-p_1(\mathbf{x}_i)) \\ &= \prod_{y_i=1} \frac{\exp(\mathbf{x_i}^\top\boldsymbol{\beta})}{1+\exp(\mathbf{x_i}^\top\boldsymbol{\beta})} \prod_{y_i=0} \frac{1}{1+\exp(\mathbf{x_i}^\top\boldsymbol{\beta})}, \end{aligned}$$

or, more simply:

$$\begin{aligned} \hat{\boldsymbol{\beta}} &= \arg\max_{\boldsymbol{\beta}}\{L(\boldsymbol{\beta})\} = \arg\max_{\boldsymbol{\beta}}\{\ln L(\boldsymbol{\beta})\} \\ &= \arg\max_{\boldsymbol{\beta}} \left\{ \sum_{y_i=1} \ln p_1(\mathbf{x}_i) + \sum_{y_i=0} \ln(1-p_1(\mathbf{x}_i)) \right\} \\ &= \ldots \text{ (terms in } \boldsymbol{\beta} \text{ and the observations } \mathbf{x}_i). \end{aligned}$$

The optimizer $\hat{\boldsymbol{\beta}}$ is then found using numerical methods; in R, the function `glm()` computes the maximum likelihood estimate directly.

Example Using the `Gapminder` data from this section's start, we obtain the following model:

```
model.LR <- glm(LE.resp ~ infant_mortality + fertility,
                data=gapminder.2011, family=binomial)
model.LR
```

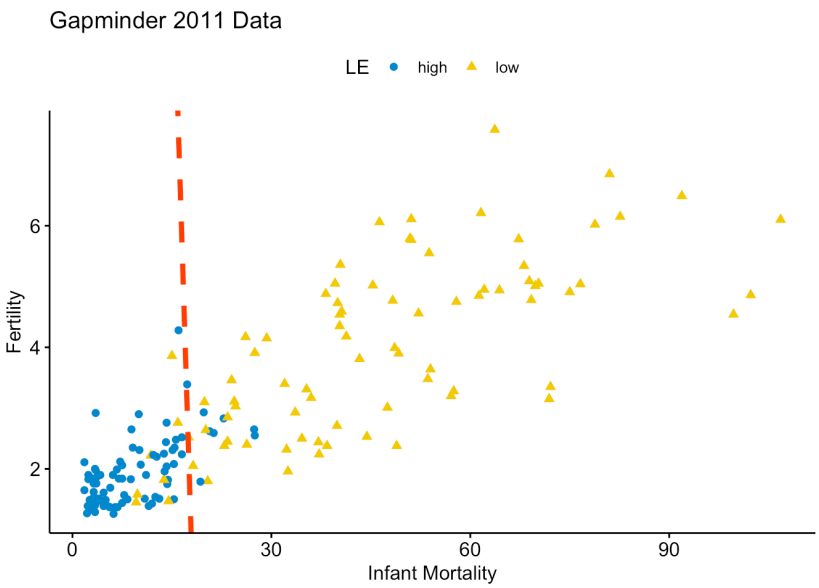
```
Coefficients:
     (Intercept)  infant_mortality         fertility
         4.58733          -0.22499           -0.06495

Degrees of Freedom: 165 Total (i.e. Null);  163 Residual
Null Deviance:      230.1
Residual Deviance: 78.17    AIC: 84.17
```

Thus

$$\hat{y}^* = \ln\left(\frac{P(Y = \texttt{high} \mid \vec{X})}{1-P(Y = \texttt{high} \mid \vec{X})}\right) = 4.59 - 0.22X_1 - 0.06X_2.$$

For a decision rule threshold of $\gamma = 0.5$, the decision boundary is shown below (compare with the linear regression boundary on page 1309).

```
beta_0 = as.numeric(model.LR[[1]][1])
beta_1 = as.numeric(model.LR[[1]][2])
beta_2 = as.numeric(model.LR[[1]][3])

slope = -beta_1/beta_2
intercept = 0.5*(1-2*beta_0)/beta_2

ggpubr::ggscatter(gapminder.2011, x="infant_mortality",
    y="fertility", shape="LE", color="LE", palette="jco",
    size = 2, xlab="Infant Mortality", ylab = "Fertility",
    title = "Gapminder 2011 Data") +
  ggplot2::geom_abline(intercept = intercept,
    slope = slope, color="red", linetype="dashed", size=1.5)
```



What is the estimated probability that the life expectancy is high in a country whose infant mortality is 15 and whose fertility is 4? By construction,

$$\begin{aligned} p_1(Y = \texttt{high} \mid X_1 = 15, X_2 = 4) &\approx g_L([1, 15, 24]^\top \hat{\boldsymbol{\beta}}) \\ &= \frac{\exp(4.59 - 0.22(15) - 0.06(4))}{1 + \exp(4.59 - 0.22(15) - 0.06(4))} \\ &= \frac{\exp(0.9526322)}{1 + \exp(0.9526322)} = 0.72. \end{aligned}$$

How does all of this square up with the statistical learning framework of Sections 19 and 20: no testing set has made an appearance, no misclassification or mean squared error rate has been calculated.

Next, we **randomly** select 116 observations, say, and train a logistic regression model on this training set Tr to obtain:

```
set.seed(0)
ind.train = sample(nrow(gapminder.2011),
    round(0.7*nrow(gapminder.2011)),replace=FALSE)
```

```
gapminder.2011.tr = gapminder.2011[ind.train,]
gapminder.2011.te = gapminder.2011[-ind.train,]

model.LR.tr <- glm(LE.resp ~ infant_mortality + fertility,
            family=binomial, data=gapminder.2011.tr)
model.LR.tr
```

```
Coefficients:
     (Intercept)  infant_mortality         fertility
          6.1194           -0.2050           -0.6653

Degrees of Freedom: 115 Total (i.e. Null);  113 Residual
Null Deviance:      159.1
Residual Deviance: 50.83    AIC: 56.83
```

Thus,

$$\hat{y}^* = 6.12 - 0.21x_1 - 0.67x_2.$$

Now, compute

$$\hat{p}_i = P(Y_i = \texttt{high} \mid X_1 = x_{1,i}, X_2 = x_{2,i}) = \frac{\exp(\hat{y}_i^*)}{1 + \exp(\hat{y}_i^*)}$$

on the observations in the testing set Te (see below).[15]

15: The observations in the original dataset *not* in Tr.

```
beta_0 = as.numeric(model.LR[[1]][1])
beta_1 = as.numeric(model.LR[[1]][2])
beta_2 = as.numeric(model.LR[[1]][3])

gapminder.2011.te$y.star = beta_0 +
    beta_1*gapminder.2011.te$infant_mortality +
    beta_2*gapminder.2011.te$fertility
gapminder.2011.te$p.1 = exp(gapminder.2011.te$y.star)/
    (1+exp(gapminder.2011.te$y.star))
gapminder.2011.te$MSE = (gapminder.2011.te$p.1 -
    gapminder.2011.te$LE.resp)^2

summary(gapminder.2011.te$LE.resp)
```

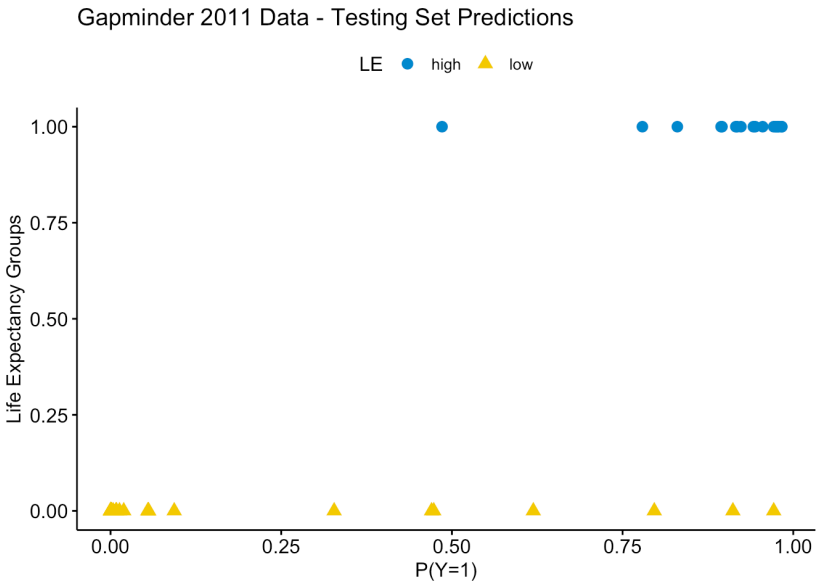
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    0.00    0.00    0.36    1.00    1.00
```

We obtain

$$\text{MSE}_{\text{Te}} = \frac{1}{50}\sum_{i=1}^{50}(\hat{p}_i - \mathcal{I}[Y_i = \texttt{high}])^2 = 0.075.$$

Is that a good test error? It is difficult to answer without more context. Perhaps a more intuitive way to view the situation is to make actual predictions and to explore their quality.

```
ggpubr::ggscatter(gapminder.2011.te, x="p.1", y="LE.resp",
    shape="LE", color="LE", palette="jco", size = 3,
    xlab="P(Y=1)", ylab = "Life Expectancy Groups",
    title = "Gapminder 2011 Data - Testing Set Predictions")
```



For $\alpha \in [0, 1]$, we further define

$$\text{pred}_i(\alpha) = \begin{cases} \texttt{high} & \text{if } \hat{p}_i > \alpha \\ \texttt{low} & \text{else} \end{cases}$$

In the specific version of Te used in this example, 36% of the nations had a `high` life expectancy.

```
gapminder.2011.te$pred81 = ifelse(gapminder.2011.te$p.1 > 0.81,
                                  1, 0)
table(gapminder.2011.te$LE.resp,gapminder.2011.te$pred81)
```

If we set $\alpha = 0.81$, then the model predicts that 36% of the test nations will have a `high` life expectancy, and the **confusion matrix** on Te is shown below:

$\alpha = 0.81$		**prediction** 0	1
actual	0	30	2
	1	2	16

But why pick $\alpha = 0.81$ instead of $\alpha = 0.5$, say?[16] In the latter case, 42% of nations are predicted to have high life expectancy, and the confusion matrix on Te is as in the next page.

16: In a sense, this could prove to be the only rational choice in the absence of information.

```
gapminder.2011.te$pred50 = round(gapminder.2011.te$p.1, 0)
table(gapminder.2011.te$LE.resp,gapminder.2011.te$pred50)
```

$\alpha = 0.5$		**prediction**	
		0	1
actual	0	28	4
	1	1	17

We will revisit this question at the end of this section (*ROC Curve*).

21.2.2 Discriminant Analysis

In logistic regression, we model $P(Y = C_k \mid \mathbf{x})$ directly *via* the logistic function

$$p_1(\mathbf{x}) = \frac{\exp(\mathbf{x}^\top \hat{\boldsymbol{\beta}})}{1 + \exp(\mathbf{x}^\top \hat{\boldsymbol{\beta}})}.$$

We have discussed some of the properties of the process in the previous section, but it should be noted that logistic regression is sometimes contra-indicated:

- when the classes are **well-separated**, the coefficient estimates may be **unstable** (adding as little as one additional point to Tr could change the coefficients substantially);
- when Tr is small and the distribution of the predictors is roughly **Gaussian** in each of the classes $Y = C_k$, the coefficient estimates may be unstable too;
- when there are more than 2 response levels, it is not always obvious how to select an extension of logistic regression.

In **discriminant analysis** (DA), we instead model

$$P(\mathbf{x} \mid Y = C_k),$$

the distribution of the predictors $\vec{X}$ conditional on the level of Y, and use Bayes' Theorem to obtain

$$P(Y = C_k \mid \mathbf{x}),$$

the probability of observing the response conditional on the predictors.

Let $\mathcal{C} = \{C_1, \ldots, C_K\}$ be the K response levels, $K \geq 2$, and denote by π_k the probability that a random observation lies in C_k, for $k \in \{1, \ldots, K\}$; π_k is the **prior**

$$\pi_k = P(Y = C_k) = \frac{|C_k|}{N}.$$

Let $f_k(\mathbf{x}) = P(\mathbf{x} \mid Y = C_k)$ be the **conditional density function** of the distribution of $\vec{X}$ in C_k; we would expect $f_k(\mathbf{x})$ to be large if there is a high probability that an observation in C_k has a corresponding predictor $\vec{X} \approx \mathbf{x}$, and small otherwise.

According to Bayes' Theorem,

$$\begin{aligned} p_k(\mathbf{x}) &= P(Y = C_k \mid \mathbf{x}) \\ &= \frac{P(\mathbf{x} \mid Y = C_k) \cdot P(Y = C_k)}{P(\mathbf{x})} \\ &= \frac{P(\mathbf{x} \mid Y = C_k) \cdot P(Y = C_k)}{P(\mathbf{x} \mid Y = C_1) \cdot P(Y = C_1) + \cdots + P(\mathbf{x} \mid Y = C_K) \cdot P(Y = C_K)} \\ &= \frac{\pi_k f_k(\mathbf{x})}{\pi_1 f_1(\mathbf{x}) + \cdots + \pi_K f_K(\mathbf{x})}. \end{aligned}$$

Given an observation $\mathbf{x} \in \text{Te}$, the DA classifier is

$$\hat{C}_{\text{DA}}(\mathbf{x}) = C_{\arg\max_j \{p_{j(\mathbf{x})}\}}.$$

In order to say more about discriminant analysis, we need to make additional assumptions on the nature of the underlying distributions.

Linear Discriminant Analysis If there is only one predictor ($p = 1$), we make the **Gaussian assumption**,

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left[-\frac{1}{2}\left(\frac{x - \mu_k}{\sigma_k}\right)^2\right],$$

where μ_k and σ_k are the mean and the standard deviation, respectively, of the predictor for all observations in class C_k.[17]

17: Any other predictor distribution could be used if it is more appropriate for Tr, and we could assume that the standard deviations or the means (or both) are identical across classes.

If we further assume that $\sigma_k \equiv \sigma$ for all k, then

$$\begin{aligned} p_k(x) &= \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x - \mu_k}{\sigma}\right)^2\right]}{\pi_1 \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x - \mu_1}{\sigma}\right)^2\right] + \cdots + \pi_K \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x - \mu_K}{\sigma}\right)^2\right]} \\ &= \frac{\pi_k \exp\left[-\frac{1}{2}\left(\frac{x - \mu_k}{\sigma}\right)^2\right]}{\pi_1 \exp\left[-\frac{1}{2}\left(\frac{x - \mu_1}{\sigma}\right)^2\right] + \cdots + \pi_K \exp\left[-\frac{1}{2}\left(\frac{x - \mu_K}{\sigma}\right)^2\right]} \\ &= \frac{\pi_k \exp\left[\frac{\mu_k}{\sigma^2}\left(x - \frac{\mu_k}{2}\right)\right] \exp\left(-\frac{x^2}{2\sigma^2}\right)}{\left\{\pi_1 \exp\left[\frac{\mu_1}{\sigma^2}\left(x - \frac{\mu_1}{2}\right)\right] + \cdots + \pi_K \exp\left[\frac{\mu_K}{\sigma^2}\left(x - \frac{\mu_K}{2}\right)\right]\right\} \exp\left(-\frac{x^2}{2\sigma^2}\right)} \\ &= \pi_k \exp\left[\frac{\mu_k}{\sigma^2}\left(x - \frac{\mu_k}{2}\right)\right] \cdot A(x). \end{aligned}$$

We do not need to compute the actual probabilities $p_k(x)$ directly if we are only interested in classification; in that case, the **discriminant score** for each class may be more useful:

$$\delta_k(x) = \ln p_k(x) = \ln \pi_k + x\frac{\mu_k}{\sigma^2} - \frac{\mu_k}{2\sigma^2} + \ln A(x).$$

As $\ln A(x)$ is the same for all k, we can drop it from the score as it does not contribute to relative differences in class scores; given an observation

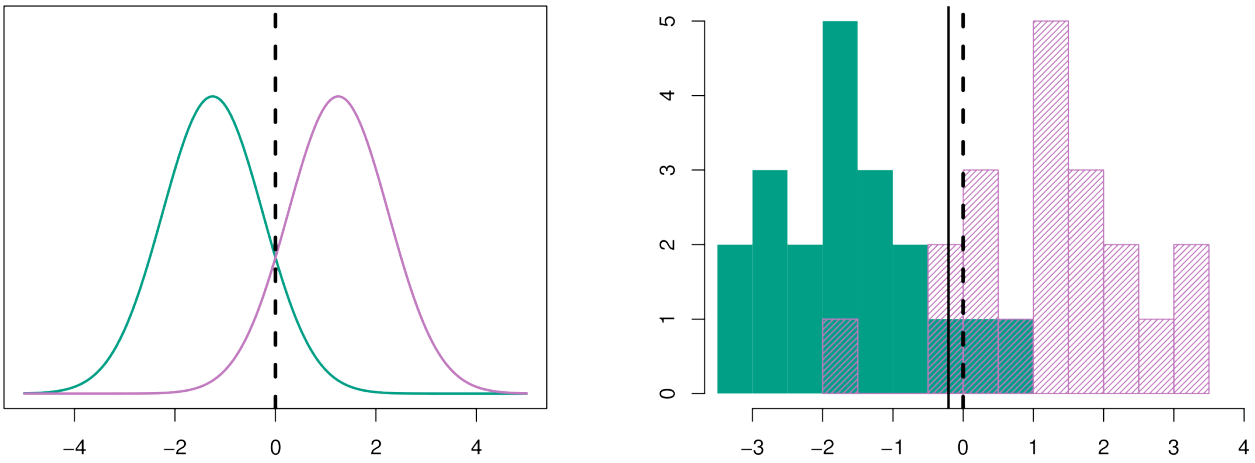


Figure 21.4: Midpoint of two theoretical normal distributions (dashed line); midpoint of two empirical normal distributions (solid line). Observations to the left of the decision boundary are classified as green, those to the right as purple. [18]

$x \in \text{Te}$, the **linear discriminant analysis** (LDA) classifier with $p = 1$ is

$$\hat{C}_{\text{LDA}}(\mathbf{x}) = C_{\arg\max_j \{\hat{\delta}_{j(\mathbf{x})}\}}.$$

Under other assumptions on the density function, the discriminant score formulation may change.

The "linear" in LDA comes from the linearity of the discriminant scores δ_k.[18]

18: After the $\ln A(x)$ term has been dropped.

If $K = 2$ and $\pi_1 = \pi_2 = 0.5$, the midpoint $x^* = \frac{1}{2}(\mu_1 + \mu_2)$ of the predictor means in C_1 and C_2 plays a crucial role. Indeed, the discriminant scores $\delta_1(x)$ and $\delta_2(x)$ meet when

$$x^* \frac{\mu_1}{\sigma^2} - \frac{\mu_1^2}{2\sigma^2} = x^* \frac{\mu_2}{\sigma^2} - \frac{\mu_2^2}{2\sigma^2} \implies x^* = \frac{\mu_1 + \mu_2}{2},$$

as long as $\mu_1 \neq \mu_2$. If $\mu_1 < \mu_2$, say, then the decision rule simplifies to

$$\hat{C}(x) = \begin{cases} C_1 & \text{if } x \leq x^* \\ C_2 & \text{if } x > x^* \end{cases}$$

The principle is illustrated in Figure 21.4.

In practice, we estimate π_k, μ_k and σ from Tr:

$$\hat{\pi}_k = \frac{N_k}{N}, \quad \hat{\mu}_k = \frac{1}{N_k} \sum_{y_i \in C_k} x_i$$

$$\hat{\sigma}^2 = \sum_{k=1}^{K} \frac{N_k - 1}{N - K} \left(\frac{1}{N_k - 1} \sum_{y_i \in C_k} (x_i - \hat{\mu}_k)^2 \right).$$

If there are $p > 1$ predictors, we can still make the Gaussian assumption, but adapted to $\mathbb{R}^p$:

$$f_k(\mathbf{x}) = \frac{1}{(2\pi)^{p/2} |\boldsymbol{\Sigma}_k|^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k), \right],$$

where $\boldsymbol{\mu}_k = (\overline{X_1}, \ldots, \overline{X_p})$ and $\boldsymbol{\Sigma}_k(j,i) = \text{Cov}(X_i, X_j)$ for all $\vec{X}$ with $Y = C_k$.

If we further assume that $\sigma_k \equiv \boldsymbol{\Sigma}$ for all k, then we can show that the discriminant score is, again, linear (in $\mathbf{x}$):

$$\delta_{k;\text{LDA}}(\mathbf{x}) = \mathbf{x}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k - \frac{1}{2} \boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k + \ln \pi_k = c_{k,0} + \mathbf{c}_k^\top \mathbf{x}.$$

We can estimate $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}$ from the data, from which we can recover the estimates

$$P(Y = C_k \mid \mathbf{x}) \approx \hat{p}_k(\mathbf{x}) = \frac{\exp(\hat{\delta}_{k;\text{LDA}}(\mathbf{x}))}{\sum_{j=1}^{K} \exp(\hat{\delta}_{j;\text{LDA}}(\mathbf{x}))}.$$

The decision rule is as before: given an observation $\mathbf{x} \in \text{Te}$, the LDA classifier with $p > 1$ is

$$\hat{C}_{\text{LDA}} = C_{\arg\max_j \{\hat{\delta}_{j;LDA}(\mathbf{x})\}}.$$

Quadratic Discriminant Analysis The assumption that the conditional probability functions be Gaussians with the same covariance in each training class may be a stretch in some situations.

If $\boldsymbol{\Sigma}_i \neq \boldsymbol{\Sigma}_j$ for at least one pair of classes (i, j), then a similar process gives rise to **quadratic discriminant analysis** (QDA), which reduces to discriminant scores

$$\begin{aligned}\delta_{k;\text{QDA}}(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) + \ln \pi_k \\ &= -\frac{1}{2}\mathbf{x}^\top \boldsymbol{\Sigma}_k^{-1}\mathbf{x} + \mathbf{x}^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k + \ln \pi_k.\end{aligned}$$

To learn the LDA model, we must estimate $Kp + \frac{p(p+1)}{2}$ parameters from Tr;[19] to learn QDA, $K\left(p + \frac{p(p+1)}{2}\right)$.[20] QDA is thus more complex (and more flexible) than LDA.

The latter is recommended if Tr is small; the former if Tr is large, but LDA will yield high bias if the $\boldsymbol{\Sigma}_k \equiv \boldsymbol{\Sigma}$ assumption is invalid. Note that LDA gives rise to nearly **linear separating hypersurfaces** and QDA to **quadratic** ones.

19: p parameters for each $\hat{\boldsymbol{\mu}}_k$ and $1 + \cdots + p$ parameters for $\hat{\boldsymbol{\Sigma}}$.

20: p parameters for each $\hat{\boldsymbol{\mu}}_k$ and $1 + \cdots + p$ parameters for each $\hat{\boldsymbol{\Sigma}}_k$.

Gaussian Naïve Bayes Classification If we assume further that each $\boldsymbol{\Sigma}_k$ is **diagonal** (that is, if we assume that the features are independent from each other in each class), we obtain the **Gaussian naïve Bayes classifier** (GNBC), with discriminant scores given by

$$\delta_{k;\text{GNBC}}(\mathbf{x}) = -\frac{1}{2}\sum_{j=1}^{p} \frac{(x_j - \mu_{k,j})^2}{\sigma_{k,j}^2} + \ln \pi_k.$$

The classification process continues as before. Note that the assumption of independence is usually not met, hence the "naïve" part in the name.

In spite of this, GNBC can prove very useful when p is too large, where both LDA and QDA break down.

Note that this approach can also be used for mixed feature vectors, by using combinations of probability mass functions and probability density functions in $f_{k,j}(x_j)$, as required. We will re-visit NBC in Section 21.4 (*Naïve Bayes Classifiers*).

Logistic Regression (Reprise) We can also recast the 2−class LDA model as

$$\ln\left(\frac{p_0(\mathbf{x})}{1-p_0(\mathbf{x})}\right) = \ln(p_0(\mathbf{x})) - \ln(p_1(\mathbf{x})) = \delta_0(\mathbf{x}) - \delta_1(\mathbf{x}) = a_0 + \mathbf{a}^\top \mathbf{x},$$

which has the same form as logistic regression.

It is not the same model, however:

- in **logistic regression**, the parameters are estimated using the maximum likelihood $P(Y \mid \mathbf{x})$;
- in **LDA**, the parameters are estimated using the full likelihood $P(\mathbf{x} \mid Y)P(\mathbf{x}) = P(\mathbf{x}, Y)$.

Example We finish this section by giving an example of LDA and QDA on the 2011 Gapminder data (we will use the same training set Tr with $N = 116$ observations and testing set Te with $M = 50$ observations).

Given an observation $\mathbf{x} \in$ Te, we use a decision rule based on the probabilities $\hat{p}_0(\mathbf{x}), \hat{p}_1(\mathbf{x})$ and a decision threshold $\alpha \in (0, 1)$. On the training set Tr, we find:

```
library(dplyr)
tmp = gapminder.2011.tr |> group_by(LE.resp) |>
    summarise(N.k=n(), mean.im=mean(infant_mortality),
    mean.f=mean(fertility))

N.0 = tmp[[2]][1]
N.1 = tmp[[2]][2]
N = N.0 + N.1

mu.0 = t(matrix(cbind(tmp[[3]][1],tmp[[4]][1])))
mu.1 = t(matrix(cbind(tmp[[3]][2],tmp[[4]][2])))
mu   = (N.0*mu.0+N.1*mu.1)/N

tmp <- gapminder.2011.tr |>
  split(gapminder.2011.tr$LE.resp) |>
  purrr::map(select, c("infant_mortality","fertility")) |>
  purrr::map(cov)

Sigma.0 <- tmp[[1]]
Sigma.1 <- tmp[[2]]
Sigma   <- cov(gapminder.2011.tr[,c("infant_mortality",
                                    "fertility")])
```

which yields

$$
\begin{aligned}
&N_0 = 51,\ N_1 = 65,\ \hat{\pi}_0 = 51/116,\ \hat{\pi}_1 = 65/116,\\
&\hat{\boldsymbol{\mu}}_0 = (45.40, 4.08)^\top,\ \hat{\boldsymbol{\mu}}_1 = (9.57, 1.92)^\top\\
&\boldsymbol{\Sigma}_0 = \begin{pmatrix} 496.51 & 23.38 \\ 23.38 & 2.17 \end{pmatrix},\ \boldsymbol{\Sigma}_1 = \begin{pmatrix} 42.79 & 2.14 \\ 2.14 & 0.31 \end{pmatrix}\\
&\hat{\boldsymbol{\mu}} = (25.30, 2.87)^\top,\ \boldsymbol{\Sigma} = \begin{pmatrix} 557.89 & 30.51 \\ 30.51 & 2.27 \end{pmatrix}
\end{aligned}
$$

We invert the matrices $\boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2, \boldsymbol{\Sigma}$ using R's `matlib::inv()`, and plug in the results in the LDA and QDA score formulas to obtain:[21]

21: Is this the best way to store the LDA/QDA functions in R?

```
gapminder.2011.te$d.0.LDA = -4.780979901 -
    0.0633308892*gapminder.2011.te$infant_mortality +
    2.645503201*gapminder.2011.te$fertility

gapminder.2011.te$d.1.LDA = -2.277022950 -
    0.1094188053*gapminder.2011.te$infant_mortality +
    2.313946886*gapminder.2011.te$fertility

gapminder.2011.te$d.0.QDA = -4.657130536 -
    0.002040555604*gapminder.2011.te$infant_mortality^2 +
    0.00614539606*gapminder.2011.te$infant_mortality +
    0.04390614038*gapminder.2011.te$infant_mortality*
        gapminder.2011.te$fertility +
    1.811698768*gapminder.2011.te$fertility -
    0.4663036203*gapminder.2011.te$fertility^2

gapminder.2011.te$d.1.QDA = -6.700309855 -
    0.01775013332*gapminder.2011.te$infant_mortality^2 -
    0.1263473930*gapminder.2011.te$infant_mortality +
    0.2427525754*gapminder.2011.te$infant_mortality*
        gapminder.2011.te$fertility +
    7.005915844*gapminder.2011.te$fertility -
    2.429442185*gapminder.2011.te$fertility^2
```

Thus,

$$
\begin{aligned}
\hat{\delta}_{0;\text{LDA}} &= -4.78 - 0.06x_1 + 2.65x_2\\
\hat{\delta}_{1;\text{LDA}} &= -2.28 - 0.11x_1 + 2.31x_2\\
\hat{\delta}_{0;\text{QDA}} &= -4.66 - 0.002x_1^2 + 0.01x_1 + 0.04x_1x_2 + 1.81x_2 - 0.47x_2^2\\
\hat{\delta}_{1;\text{QDA}} &= -6.70 - 0.02x_1^2 - 0.13x_1 + 0.24x_1x_2 + 7.01x_2 - 2.43x_2^2
\end{aligned}
$$

With the class probability estimates

$$
\begin{aligned}
\hat{p}_{1;\text{LDA}} &= \frac{\exp(\hat{\delta}_{1;\text{LDA}})}{\exp(\hat{\delta}_{0;\text{LDA}}) + \exp(\hat{\delta}_{1;\text{LDA}})},\\
\hat{p}_{1,\text{QDA}} &= \frac{\exp(\hat{\delta}_{1;\text{QDA}})}{\exp(\hat{\delta}_{0;\text{QDA}}) + \exp(\hat{\delta}_{1;\text{QDA}})}
\end{aligned}
$$

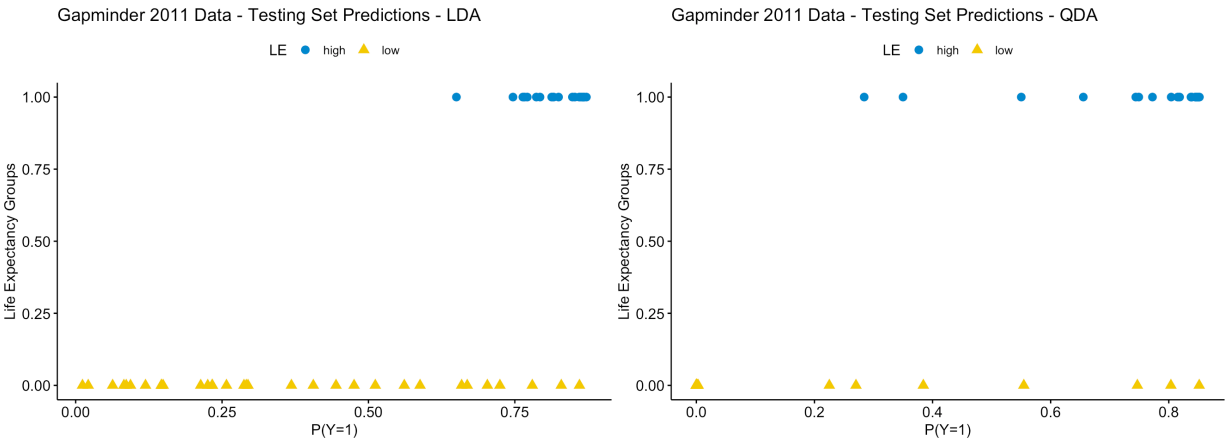
```
gapminder.2011.te$p.1.LDA = exp(gapminder.2011.te$d.1.LDA)/
    (exp(gapminder.2011.te$d.0.LDA)+exp(gapminder.2011.te$d.1.LDA))
gapminder.2011.te$p.1.QDA = exp(gapminder.2011.te$d.1.QDA)/
    (exp(gapminder.2011.te$d.0.QDA)+exp(gapminder.2011.te$d.1.QDA))
```

and the decision threshold set at $\alpha = 0.5$, the LDA and QDA life expectancy classifiers are defined on Te by

$$\hat{C}_{\alpha;\text{LDA}}(\mathbf{x}) = \begin{cases} 1\,(\texttt{high}) & \text{if } p_{1;\text{LDA}}(\mathbf{x}) \geq 0.5 \\ 0\,(\texttt{low}) & \text{else} \end{cases}$$

$$\hat{C}_{\alpha;\text{QDA}}(\mathbf{x}) = \begin{cases} 1\,(\texttt{high}) & \text{if } p_{1;\text{QDA}}(\mathbf{x}) \geq 0.5 \\ 0\,(\texttt{low}) & \text{else} \end{cases}$$

```
ggpubr::ggscatter(gapminder.2011.te, x="p.1.LDA", y="LE.resp",
    shape="LE", color="LE", palette="jco", size = 3,
    xlab="P(Y=1)", ylab = "Life Expectancy Groups",
    title = "Gapminder 2011 Data - Test Predictions - LDA")

ggpubr::ggscatter(gapminder.2011.te, x="p.1.QDA", y="LE.resp",
    shape="LE", color="LE", palette="jco", size = 3,
    xlab="P(Y=1)", ylab = "Life Expectancy Groups",
    title = "Gapminder 2011 Data - Test Predictions - QDA")
```



The $\alpha = 0.5$ confusion matrices for the LDA and QDA classifiers are:

```
LDA.table = function(x,alpha){
  tmp = ifelse(x$p.1.LDA > alpha, 1, 0)
  LDA.table = table(factor(x$LE.resp, levels = 0:1),
                    factor(tmp, levels = 0:1)) }

QDA.table = function(x,alpha){
  tmp = ifelse(x$p.1.QDA > alpha, 1, 0)
  QDA.table = table(factor(x$LE.resp, levels = 0:1),
                    factor(tmp, levels = 0:1)) }
```

```
test.LDA=LDA.table(gapminder.2011.te,0.5)
test.QDA=QDA.table(gapminder.2011.te,0.5)
```

LDA $\alpha = 0.5$		**prediction** 0	1	QDA $\alpha = 0.5$		**prediction** 0	1
actual	0	22	10	**actual**	0	28	4
	1	0	18		1	2	16

In the LDA case, we further have

- accuracy = $\frac{22+18}{22+10+0+18} = 80\%$
- misclassification rate = $\frac{10+0}{22+10+0+18} = 20\%$
- FPR = $\frac{0}{0+18} = 0\%$
- FNR = $\frac{10}{22+10} = 31.25\%$
- TPR = $\frac{22}{22+10} = 68.75\%$
- TNR = $\frac{18}{0+18} = 100\%$

In the QDA case:

- accuracy = $\frac{28+16}{28+4+2+16} = 88\%$
- misclassification rate = $\frac{4+2}{28+4+2+16} = 12\%$
- FPR = $\frac{2}{2+16} = 11.1\%$
- FNR = $\frac{4}{28+4} = 12.5\%$
- TPR = $\frac{28}{28+4} = 87.5\%$
- TNR = $\frac{16}{2+16} = 88.9\%$

At first glance, it would certainly seem that the QDA model performs better (at a decision threshold of $\alpha = 0.5$), but the FPR is not ideal. What would be the ideal value of α? How would we find it?

21.2.3 ROC Curve

The **receiver operating characteristic** (ROC) curve plots the true positive rate against the false positive rate for classifiers obtained by varying the decision threshold α in $[0, 1]$. The important realization is that a classifier that is completely random would lie on the line TPR = FPR. Thus, the **ideal threshold** would be the one associated with the model which is **farthest** from that line.

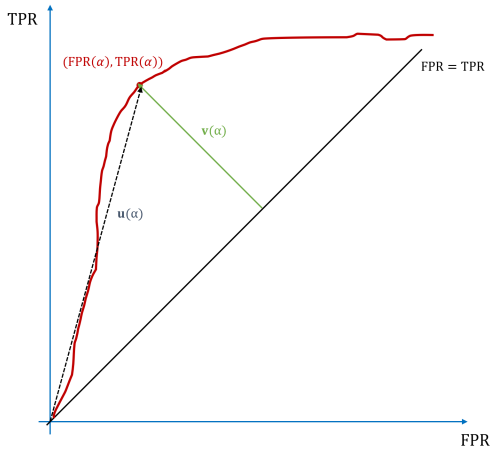


Figure 21.5: Illustration of ROC curve concepts.

Let $\mathbf{u}(\alpha)$ be the vector from $\mathbf{0}$ to the $(\text{FPR}(\alpha), \text{TPR}(\alpha))$ coordinates of the classifier with threshold α, and let $\mathbf{v}(\alpha)$ be the vector through $(\text{FPR}(\alpha), \text{TPR}(\alpha))$ and perpendicular to the line TPR = FPR. The ideal α^* satisfies:

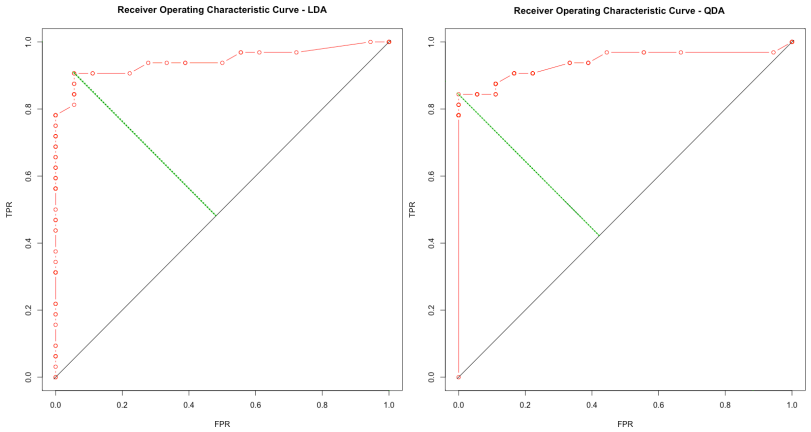
$$\begin{aligned}
\alpha^* &= \arg\max_{\alpha}\{\|\mathbf{v}(\alpha)\|\} \\
&= \arg\max_{\alpha}\{\|\mathbf{v}(\alpha)\|^2\} \\
&= \arg\max_{\alpha}\left\{\left\|\mathbf{u}(\alpha) - \text{proj}_{(1,1)}\mathbf{u}(\alpha)\right\|^2\right\} \\
&= \arg\max_{\alpha}\left\{\left\|(\text{FPR}(\alpha), \text{TPR}(\alpha)) - \text{proj}_{(1,1)}(\text{FPR}(\alpha), \text{TPR}(\alpha))\right\|^2\right\} \\
&= \arg\max_{\alpha}\left\{\|(\text{FPR}(\alpha) - \text{TPR}(\alpha), \text{TPR}(\alpha) - \text{FPR}(\alpha))\|^2\right\} \\
&= \arg\max_{\alpha}\{(\text{FPR}(\alpha) - \text{TPR}(\alpha))^2\}.
\end{aligned}$$

Example For the LDA and QDA classifiers built with the 2011 Gapminder data (see preceding section), the false positive rates (FPR), false negative rates (FNR), true positive rates (TPR), true negative rates (TNR), and misclassification rates (MCR) when the decision threshold α varies from 0.01 to 0.99 by steps of length 0.01 are computed below:

```
fpr=c()
fnr=c()
tpr=c()
tnr=c()
mcr=c()

for(alpha in 1:99){
  tmp=LDA.table(gapminder.2011.te,alpha/100)
  mcr[alpha]=(tmp[1,2]+tmp[2,1])/sum(tmp)
  fpr[alpha]=tmp[2,1]/(tmp[2,1]+tmp[2,2])
  fnr[alpha]=tmp[1,2]/(tmp[1,1]+tmp[1,2])
  tnr[alpha]=tmp[2,2]/(tmp[2,1]+tmp[2,2])
  tpr[alpha]=tmp[1,1]/(tmp[1,1]+tmp[1,2]) }

plot(c(0,fpr),c(0,tpr), type = "b", pch = 21,
     col = "red", xlim=c(0,1), ylim=c(0,1),
     xlab="FPR",ylab="TPR",
     main="Receiver Operating Characteristic Curve - LDA")
abline(0,1)
(index=which((fpr-tpr)^2==max((fpr-tpr)^2)))
abline(fpr[index[1]]+tpr[index[1]],-1, col="green")

for(alpha in 1:99){
  tmp=QDA.table(gapminder.2011.te,alpha/100)
  mcr[alpha]=(tmp[1,2]+tmp[2,1])/sum(tmp)
  fpr[alpha]=tmp[2,1]/(tmp[2,1]+tmp[2,2])
  fnr[alpha]=tmp[1,2]/(tmp[1,1]+tmp[1,2])
  tnr[alpha]=tmp[2,2]/(tmp[2,1]+tmp[2,2])
  tpr[alpha]=tmp[1,1]/(tmp[1,1]+tmp[1,2]) }
```

```
plot(c(0,fpr),c(0,tpr), type = "b", pch = 21,
     col = "red", xlim=c(0,1), ylim=c(0,1),
     xlab="FPR",ylab="TPR",
     main="Receiver Operating Characteristic Curve - QDA")
abline(0,1)
(index=which((fpr-tpr)^2==max((fpr-tpr)^2)))
abline(fpr[index[1]]+tpr[index[1]],-1, col="green")
```

```
[1] 73 74
[1] 28
```



In both frameworks, a number of models have identical (FPR, TPR) coordinates. With the LDA model, the ideal threshold is $\alpha^*_{\text{LDA}} = 0.73$ (coordinates $(0.056, 0.906)$); with the QDA model, it is $\alpha^*_{\text{QDA}} = 0.28$ (coordinates $(0, 0.844)$).

The corresponding confusion matrices are shown below.

LDA $\alpha^*_{\text{LDA}} = 0.73$		**prediction** 0	1	QDA $\alpha^*_{\text{QDA}} = 0.28$		**prediction** 0	1
actual	0	29	3	**actual**	0	27	5
	1	1	17		1	0	18

Which model is best? It depends on the context of the task, and on the consequences of the choice. What makes the most sense here? Is there a danger of overfitting? Is parameter tuning acceptable, from a data massaging perspective? What effect does the choice of priors have?

While we can find the **optimal** α according to the procedure highlighted above, there is another aspect of the ROC curve that may be of interest: in general, the larger the area under the curve is, the better the model may behave for non-optimal decision thresholds.

The metric is known as ROC AUC; technically, it varies between 0 and 1, but we since a classifier that is wrong more often than expected indirectly provides a classifier that is right more often than expected, [22] we focus instead on the area between the curve and the line TPR = FPR.

22: Simply predict the opposite of what it predicts.

21.3 Rare Occurrences

Before we continue and discuss other supervised approaches, we briefly touch on the problem of **rare occurrences** (or **unbalanced** dataset). Say we are looking to detect fraudulent transactions. We can build classifiers to approach this task using any number of methods, but there is a potential problem. If $(100 - \varepsilon)\%$ of observations belong to the **normal** category, and $\varepsilon\%$ to the **special** category, the model that predicts that EVERY observation is normal has $(100 - \varepsilon)\%$ accuracy.

In practice, the vast majority of transactions are legitimate, so that $0 < \varepsilon \ll 1$; the model in question has tremendous accuracy, even if it misses the point of the exercise altogether.

There are two general approaches to overcome this issue: either we **modify the algorithms** to take into account the asymmetric cost of making a classification error (through so-called **cost-sensitive classifiers** or **one-class models**), or we **modify the training data** to take into account the imbalance in the data. In the former case, we invite you to read the documentation of the methods that interest you to see how this could be achieved.

In the latter case, we could try to obtain more training data. This is the simplest method, but it is not always possible to do so, and it could be that the new data would follow the same pattern as the original data, which would leave us no better off than we were to start with.

Another alternative is to create a new training set by either **undersampling** the majority class(es) or **oversampling** the under-represented class(es). We assume for now that there are only two classes in the data (the strategies can easily be adapted to multi-class problems).

Undersampling Let $\text{Tr} = L \sqcup M_c$, where L consists of all observations in the majority case, and M_c of all observations in the minority case; by assumption, $|L| \gg |M_c|$ (and $L \cap M_c = \varnothing$). Split L into K subsets $L_1, \ldots, L_K$, each roughly of the same size; K should be selected so that $|M_c| \ll |L_i|$, for all i (in other words, even though $|L_i|$ could be larger than $|M_c|$, it is not going to be substantially so).

We then construct K training sets

$$\text{Tr}_1 = L_1 \sqcup M_c, \ldots, \text{Tr}_K = L_K \sqcup M_c;$$

for all $1 \leq i \leq K$, we train a classifier C_i (using a given algorithm) on Tr_i. Once that is done, we combine the predictions using bagging or other ensemble learning methods (see Section 21.5).

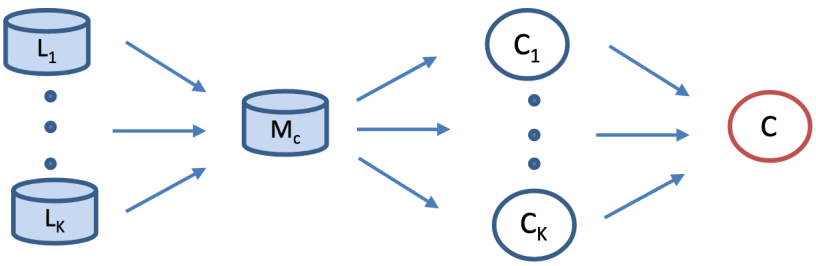


Figure 21.6: Illustration of undersampling [author unknown].

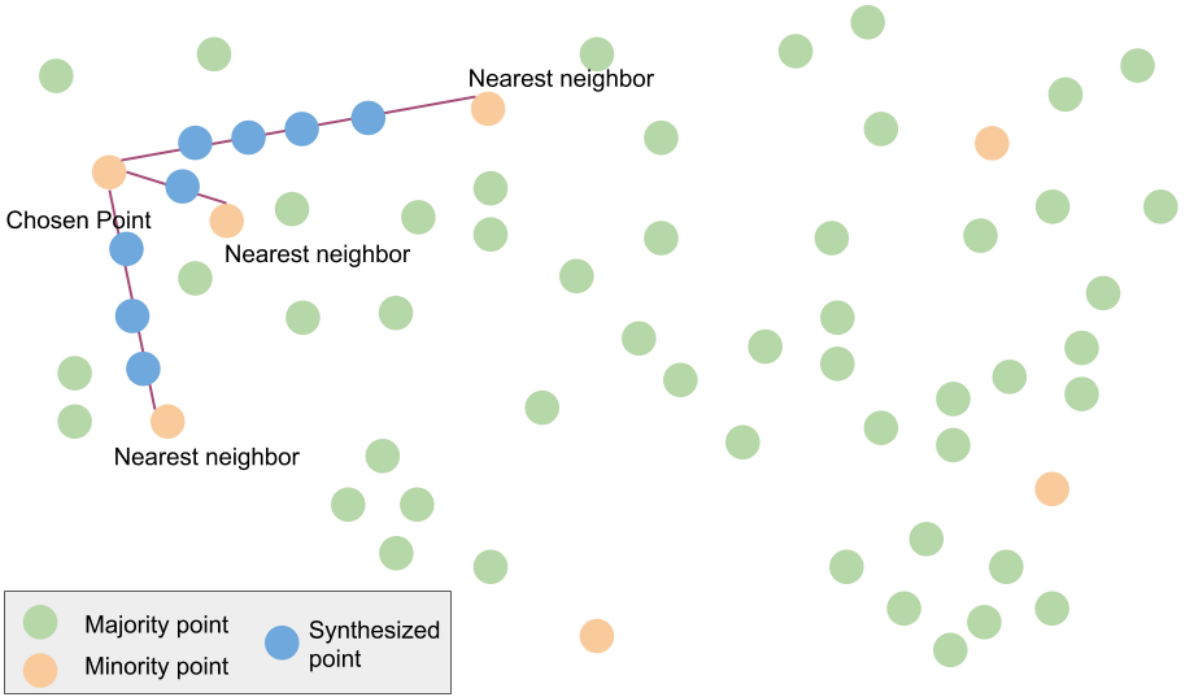


Figure 21.7: Illustration of oversampling (SMOTE) [author unknown].

Oversampling We can oversample the minority cases to create balanced datasets, but that introduces a dependency in the data that can have far-reaching effect when it comes to bias and variability.

Synthetic Minority Oversampling Technique (SMOTE) is a common approach which creates "synthetic" examples rather than oversampling with replacement – the same idea is used to create samples for handwriting recognition by perturbing training data (e.g., rotating, skewing, etc.) [5]:

1. select random integers $k \ll \ell$;
2. draw a random sample $\mathcal{V}_\ell$ of size ℓ from the minority class M_c;
3. for each $\mathbf{x} \in \mathcal{V}_\ell$, find the k nearest neighbors of $\mathbf{x}$ in M_c, say $\mathbf{z}_{\mathbf{x},1}, \ldots, \mathbf{z}_{\mathbf{x},k}$;
4. compute the vectors $\mathbf{v}_{\mathbf{x},1}, \ldots, \mathbf{v}_{\mathbf{x},k}$, originating from $\mathbf{x}$ and ending at each of the $\mathbf{z}_{\mathbf{x},1}, \ldots, \mathbf{z}_{\mathbf{x},k}$;
5. draw random values $\gamma_1, \ldots, \gamma_k \sim \mathcal{U}(0,1)$, and multiply $\mathbf{v}_{\mathbf{x},i}$ by γ_i, for each $1 \leq i \leq k$;
6. the points found at $\mathbf{x} + \gamma_i \mathbf{v}_{\mathbf{x},i}$, $1 \leq i \leq k$, are added to the set M_c.

This procedure is repeated until $|M_c| \not\ll |L|$ (see Figure 21.7).

There are variants, where we always use the same $k, \ell, \mathcal{V}_\ell$, or where we only pick one of the k nearest neighbours, etc. In general, SMOTE increases **recall**, but it comes at the cost of lower **precision**.

We will have more to say about rare occurrences in Chapter 26, *Anomaly Detection and Outlier Analysis*.

21.4 Other Supervised Approaches

In this section, we present a number of **non-parametric** approaches, with a focus on classification methods (although we will also discuss regression problems):

- **classification and regression trees** (CART) [2, 18, 14, 25];
- **support vector machines** (SVW) [2, 18, 14, 15, 9];
- **artificial neural networks** (ANN) [6, 12, 1], and
- **naïve Bayes classification** (NBC).

21.4.1 Tree-Based Methods

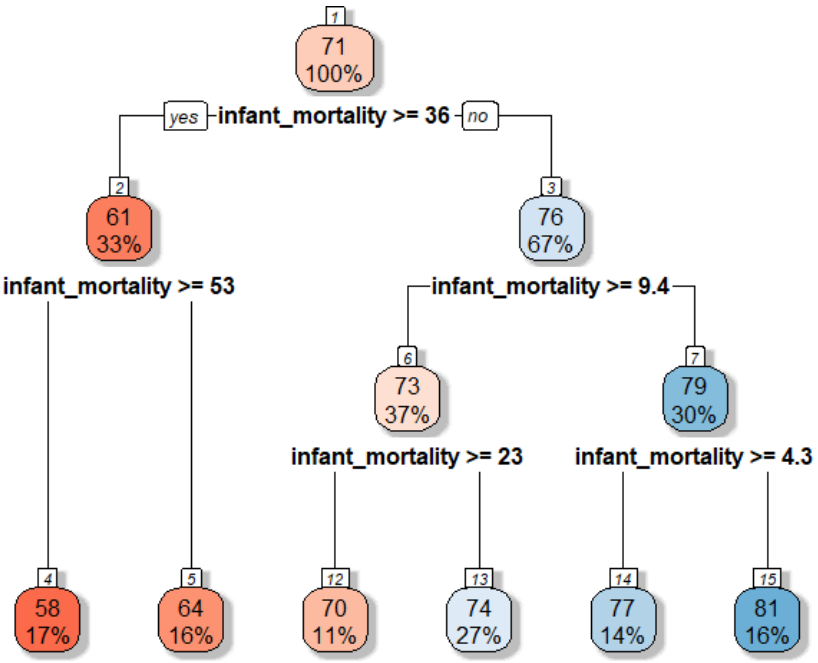
This family of methods involves **stratifying** or **segmenting** the predictor space into a small number of "simple" regions.

The set of **splitting rules** used to segment the space can be summarized using a **tree**, whence their name. Tree-based methods are **simple** and **easy to interpret**; but they don't tend to be competitive with the best supervised learning methods when it comes to predictive accuracy.

Nevertheless, there are instances when the ease of interpretability overrules the lessened accuracy. Tree-based methods are applicable both to regression and to classification problems.

Regression Trees We introduce the important concepts via the 2011 Gapminder dataset.[23] In the figure on page 1308, we saw that when X_1 and X_2 are both high, Y is low, and when X_1 and X_2 are both low, Y is high. But what is the pattern "in the middle"?

Below, we see a possible **regression tree** for the (full) dataset ($N = 166$ observations).



23: The response Y is once again the life expectancy of nations, and the predictors X_1 and X_2 are the fertility rates and infant mortality rates per nation.

The tree can also be displayed as:

```
1) root (166) 70.82349
   2) infant_mortality>=35.65 (54) 60.85370
     4) infant_mortality>=52.9 (28) 58.30714 *
     5) infant_mortality< 52.9 (26) 63.59615 *
   3) infant_mortality< 35.65 (112) 75.63036
     6) infant_mortality>=9.35 (62) 72.89516
      12) infant_mortality>=22.85 (18) 69.50000 *
      13) infant_mortality< 22.85 (44) 74.28409 *
     7) infant_mortality< 9.35 (50) 79.02200
      14) infant_mortality>=4.25 (23) 76.86087 *
      15) infant_mortality< 4.25 (27) 80.86296 *
```

Node 1 is the tree's **root** (initial node) with 166 (100%) observations; the average life expectancy for these observations is 70.82.

The root is also the tree's first **branching point**, separating the observations into two groups: node 2 with 54 observations (33%), given by "infant mortality ≥ 35.65", for which the average life expectancy is 60.85, and node 3 with 112 observations (67%), given by "infant mortality < 35.65", for which the average life expectancy is 75.63.

Note that $54 + 112 = 166$ and that

$$\frac{54(60.81) + 112(75.63)}{54 + 112} = 70.82.$$

Node 2 is an **internal node** – it is further split into two groups: node 4 with 28 observations (17%), given with the additional rule "infant mortality ≥ 52.9", for which the average life expectancy is 58.31, and node 5 with 26 observations (16%), given with the additional rule "infant mortality < 52.9", for which the average life expectancy is 63.60.

Note that $28 + 26 = 54$ and that

$$\frac{28(58.31) + 26(63.60)}{28 + 26} = 60.85.$$

Both nodes 4 and 5 are **leaves** (final nodes, terminal nodes); the tree does not grow any further on that branch.

The tree continues to grow from node 3, eventually leading to 4 leaves on that branch (there are intermediate branching points). There are 6 leaves in total, 5 branching points (including the root) and the tree's **depth** is 3 (excluding the root).

Only one feature is used in the regression tree in this example: to make a prediction for a new observation, only infant mortality is needed. If it was 21, say, the observation's leaf would be node 13 and we would predict that the life expectancy of that nation would be 74.28.

The tree diagram is a useful heuristic, especially since it allows the results to be displayed without resorting to a multi-dimensional chart, but it obscures the **predictor space's stratification**.

We can also write

$$
\begin{aligned}
R_4 &= \{(\text{infant mortality}, \text{fertility}) \mid \text{infant mortality} \geq 52.9\} \\
R_5 &= \{(\text{infant mortality}, \text{fertility}) \mid 36.65 \leq \text{infant mortality} < 52.9\} \\
R_{12} &= \{(\text{infant mortality}, \text{fertility}) \mid 22.85 \leq \text{infant mortality} < 35.65\} \\
R_{13} &= \{(\text{infant mortality}, \text{fertility}) \mid 9.35 \leq \text{infant mortality} < 22.85\} \\
R_{14} &= \{(\text{infant mortality}, \text{fertility}) \mid 4.25 \leq \text{infant mortality} < 9.35\} \\
R_{15} &= \{(\text{infant mortality}, \text{fertility}) \mid \text{infant mortality} < 4.25\}
\end{aligned}
$$

It turns out that only infant mortality is involved in the definition of the tree's **terminal nodes**. The regions are shown below:

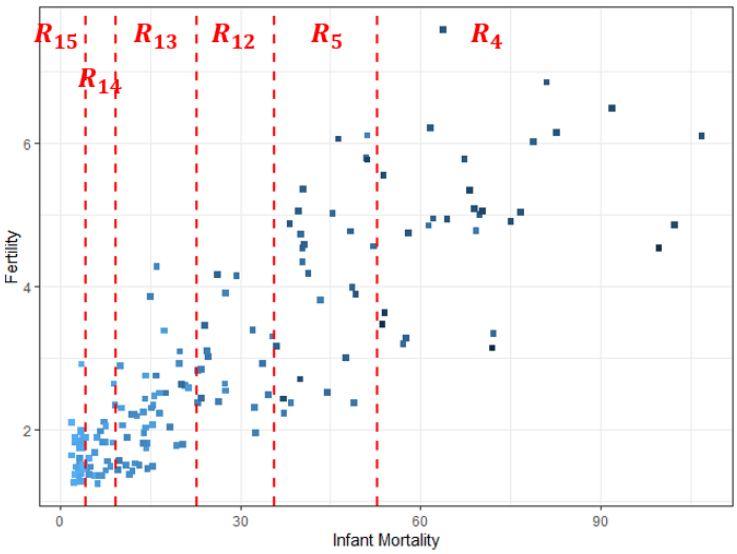


Figure 21.8: Stratification of the predictor space for the 2011 Gapminder data regression tree.

The regression tree model for life expectancy would thus be

$$
\hat{y}_i = f(\mathbf{x}_i) = \text{Avg}\{y \mid \mathbf{x} \in R_{j(i)}\} = \begin{cases} 58.3, & j(i) = 4 \\ 63.6, & j(i) = 5 \\ 69.5, & j(i) = 12 \\ 74.3, & j(i) = 13 \\ 76.9, & j(i) = 14 \\ 80.9, & j(i) = 15 \end{cases}
$$

where $R_{j(i)}$ is the region in which $\mathbf{x}_i$ falls. This tells us that infant mortality is the most important factor in determining life expectancy, with a negative correlation.[24] But it is not the only way to stratify the data: how is it an **optimal** tree?[25]

Building A Regression Tree The process is quite simple:

1. divide the predictor space $\mathcal{X} \subseteq \mathbb{R}^p$ into a disjoint union of J regions:

$$\mathcal{X} = R_1 \sqcup \cdots \sqcup R_J;$$

2. for any $\mathbf{x} \in R_j$,

$$\hat{y}(\mathbf{x}) = \text{Avg}\{y(\mathbf{z}) \mid \mathbf{z} \in R_j \cap \text{Tr}\}.$$

24: This interpretation is, of course, a coarse oversimplification, but it highlights the advantage of using a regression tree when it comes to **displaying**, **interpreting**, and **explaining** the results.

25: Recall that all supervised learning tasks are optimization problems.

The second step tells us that trees are **locally constant**.[26]

26: It would be possible to define trees that are not locally constant, if required.

In theory, the regions R_j could have any shape as long as they form a **disjoint cover** of $\mathcal{X}$; in practice, we use **hyperboxes** with $p-1$-dimensional affine boundaries that are perpendicular/parallel to the p hyperplanes $X_1 \dots \hat{X}_k \dots X_p$, $k = 1, \dots, p$.

We find the optimal $(R_1, \dots, R_J)$ by minimizing

$$\text{SSE} = \sum_{j=1}^{J} \sum_{\mathbf{x}_i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where $\hat{y}_{R_j}$ is the mean response of y in $R_j \cap \text{Tr}$. In an ideal world, we would compute SSE for all partitions of $\mathcal{X}$ into hyperboxes, and pick the one that minimizes SSE, but that is not computationally feasible, in general.

Instead, we use a growth algorithmic approach known as **recursive binary splitting**, which is both **top-down** (starts at the root and successively splits $\mathcal{X}$ *via* 2 new branches down the tree) and **greedy** (at each step of the splitting process, the best choice is made there and now, rather than by looking at long-term consequences).

Regression Tree Algorithm The algorithm has 10 steps, but it is fairly straightforward.

1. Let $\hat{y}_0 = \text{Avg}\{y(\mathbf{x}_i) \mid i = 1, \dots, N \text{ and } \mathbf{x}_i \in \text{Tr}\}$.
2. Set the baseline $\text{SSE}_0 = \sum_{i=1}^{N} (y_i - \hat{y}_0)^2$.
3. For each $1 \le k \le p$, order the predictor values of X_k in Tr: $\min_{1 \le i \le N}\{x_{i,k}\} = v_{k,1} \le v_{k,2} \le \cdots \le v_{k,N} = \max_{1 \le i \le N}\{x_{i,k}\}$.
4. For each X_k, set $s_{k,\ell} = \frac{1}{2}(v_{k,\ell} + v_{k,\ell+1})$, $\ell = 1, \dots, N-1$.
5. For each $k = 1, \dots, p$, $\ell = 1, \dots, N-1$, define

   $$R_1(k,\ell) = \{\vec{X} \in \mathbb{R}^p \mid X_k < s_{k,\ell}\}, \quad R_2(k,\ell) = \{\vec{X} \in \mathbb{R}^p \mid X_k \ge s_{k,\ell}\}.$$

   Note that $\mathcal{X} = R_1(k,\ell) \sqcup R_2(k,\ell)$ for all k, ℓ.
6. For each $k = 1, \dots, p$, $\ell = 1, \dots, N-1$, set

   $$\text{SSE}_1^{k,\ell} = \sum_{m=1}^{2} \sum_{\vec{X}_i \in R_m(k,\ell)} (y_i - \hat{y}_{R_m(k,\ell)})^2,$$

   where $\hat{y}_{R_m(k,\ell)} = \text{Avg}\{y(\mathbf{x}) \mid \mathbf{x} \in \text{Tr} \cap R_m(k,\ell)\}$.
7. Find k^*, ℓ^* for which $\text{SSE}_1^{k,\ell}$ is **minimized**.
8. Define the **children sets** $R_1^L = R_1(k^*, \ell^*)$ and $R_1^R = R_2(k^*, \ell^*)$.
9. While some children sets R_μ^ν still do not meet a **stopping criterion**, repeat steps 3 to 8, searching and minimizing SSE over $\mathcal{X} \cap R_\mu^\nu$, and producing a binary split $R_{\mu+1}^L, R_{\mu+1}^R$.[27]
10. Once the stopping criterion is met for all children sets, the tree's growth ceases, and $\mathcal{X}$ has been partitioned into J regions (renumbering as necessary)

    $$\mathcal{X} = R_1 \sqcup \cdots \sqcup R_J,$$

    on which the regression tree predicts the J responses $\{\hat{y}_1, \dots, \hat{y}_J\}$, according to $\hat{y}_j = \text{Avg}\{y(\mathbf{x}) \mid \mathbf{x} \in R_j\}$.

27: Multiple stopping criteria are used in practice, such as insisting that all final nodes contain 10 or fewer observations, etc.

For instance, if the training set was $\text{Tr} = \{(x_{1,i}, x_{2,i}, y_i)\}_{i=1}^N$, the algorithm might provide the regression tree in Figure 21.9.

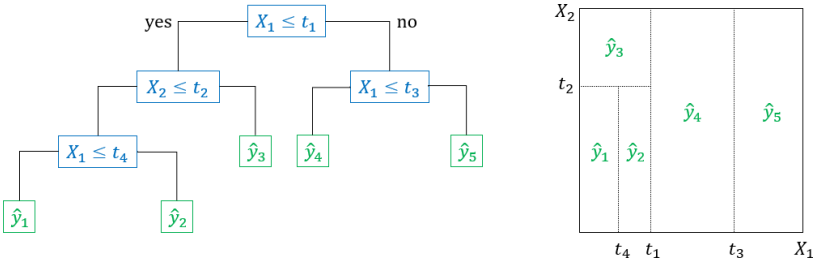


Figure 21.9: Generic recursive binary partition regression tree for a two-dimensional predictor space, with 5 leaves.

In R, the recursive binary partition algorithm is implemented in package `rpart`'s function `rpart()`.

Tree Pruning Regression trees grown with the algorithm are prone to overfitting; they can provide good predictions on Tr, but they usually make shoddy predictions on Te.[28]

A smaller tree with fewer splits might lead to lower variance and better interpretability, at the cost of a little bias. Instead of simply growing a tree T_0 until each leaf contains at most M observations, say,[29] it could be beneficial to **prune** it in order to obtain an **optimal subtree**.

We use **cost complexity pruning** (CCP) to build a sequence of candidate subtrees indexed by the complexity parameter $\alpha \ge 0$. For each such α, find a subtree $T_\alpha \subseteq T_0$ which minimizes

$$\text{SSE} + \text{complexity penalty} = \sum_{m=1}^{|T|} \sum_{\mathbf{x}_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|,$$

where $|T|$ represents the number of final nodes in T; when α is large, it is costly to have a complex tree.

This is similar to the bias-variance trade-off or the regularization framework: a good tree balances considerations of fit and complexity.

Pruning Algorithm Assume that a recursive binary splitting regression tree T_0 has been grown on Tr, using a given stopping criterion:

1. apply CCP to T_0 to obtain a "sequence" T_α of subtrees of T_0;
2. divide Tr into K folds;
3. for all $k = 1, \ldots, K$, build a regression tree $T_{\alpha;k}$ on $\text{Tr} \setminus \text{Fold}_k$ and evaluate

$$\widehat{\text{MSE}}(\alpha) = \text{Avg}_{1 \le k \le K} \{\text{MSE}_k(\alpha) \text{ of } T_{\alpha;k} \text{ on } \text{Fold}_k\};$$

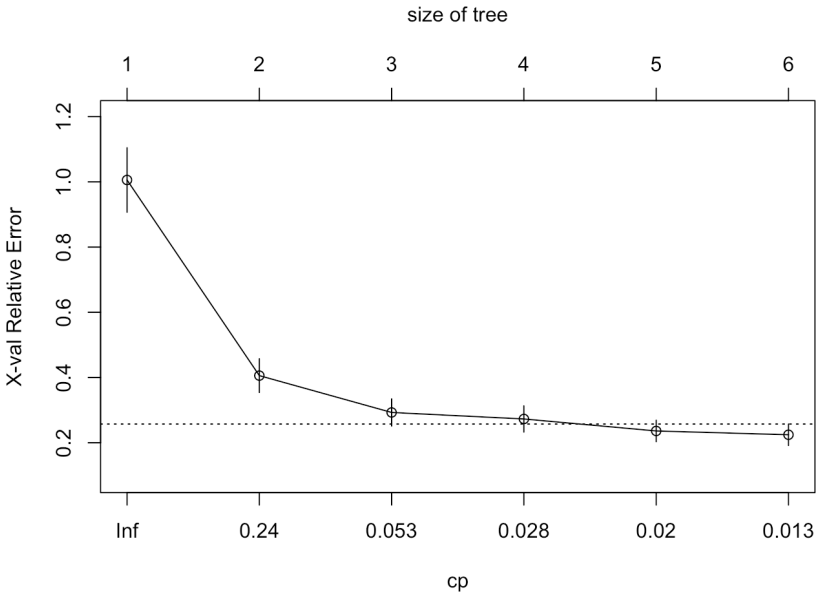
4. return T_{α^*} from step 1, where $\alpha^* = \arg\min_\alpha \{\widehat{\text{MSE}}(\alpha)\}$.

The Gapminder 2011 tree is pruned in the Figures below, using the `rpart` functions `plotcp()` (the complexity parameter α is denoted by `cp` in the code below) and `rpart()`.

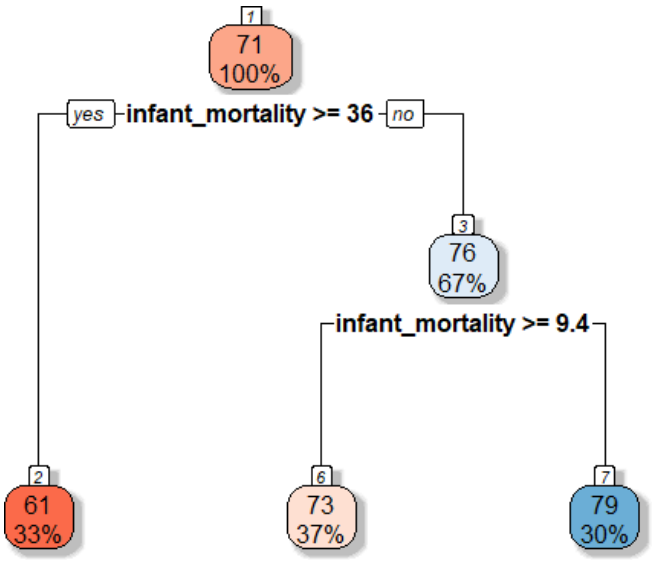
28: Because the resulting tree might be too complex – it captures noise as well as the signal.

29: Or whatever other stopping criterion might be appropriate.

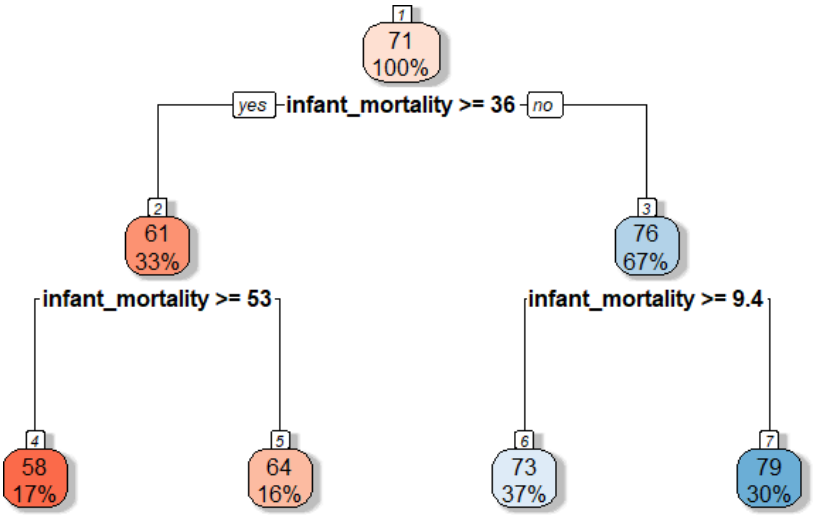
```
rpart::plotcp(reg.tree.1)
```



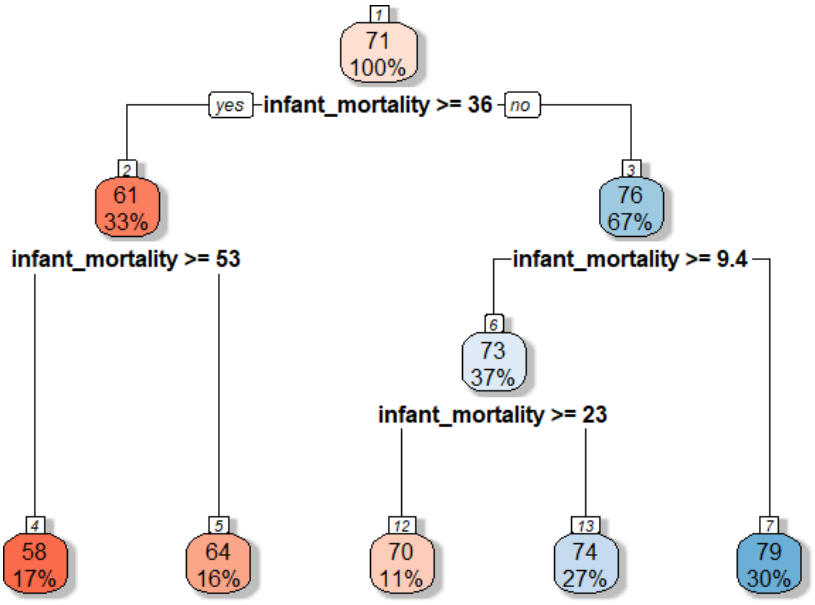

```
reg.tree.1.pruned.2 <- rpart::rpart(life_expectancy ~
    fertility + infant_mortality,
    data=gapminder.2011, cp=0.06)
rpart.plot::rpart.plot(reg.tree.1.pruned.2,
    box.palette="RdBu", shadow.col="gray", nn=TRUE)
```




```
reg.tree.1.pruned.3 <- rpart::rpart(life_expectancy ~
    fertility + infant_mortality,
    data=gapminder.2011, cp=0.028)
rpart.plot::rpart.plot(reg.tree.1.pruned.3,
    box.palette="RdBu", shadow.col="gray", nn=TRUE)
```




```
reg.tree.1.pruned.4 <- rpart::rpart(life_expectancy ~
    fertility + infant_mortality,
    data=gapminder.2011, cp=0.02)
rpart.plot::rpart.plot(reg.tree.1.pruned.4,
    box.palette="RdBu", shadow.col="gray", nn=TRUE)
```



We plotted the complexity pruning parameter, and the pruned trees for `cp` = 0.06, `cp` = 0.028, and `cp` = 0.02 in the Gapminder 2011 example. Note that the tree's complexity increases when `cp` decreases.

Classification Trees The approach for classification is much the same, with a few appropriate substitutions:

1. prediction in a terminal node is either the **class label mode** or the **relative frequency** of the class labels;

2. SSE must be replaced by some other fit measure:
   - the **classification error rate**:
   $$E = \sum_{j=1}^{J} (1 - \max_k \{\hat{p}_{j,k}\}),$$
   where $\hat{p}_{j,k}$ is the proportion of Tr observations in R_j of class k (this measure is not a recommended choice, however);
   - the **Gini index**, which measures the total variance across classes
   $$G = \sum_{j=1}^{J} \sum_k \hat{p}_{j,k}(1 - \hat{p}_{j,k}),$$
   which should be small when the nodes are **pure** ($\hat{p}_{j,k} \approx 0$ or 1 throughout the regions), and
   - the **cross-entropy deviance**
   $$D = -\sum_{j=1}^{J} \sum_k \hat{p}_{j,k} \ln \hat{p}_{j,k},$$
   which behaves like the Gini index, numerically.

One thing to note is that **classification and regression trees** (jointly known as CART) suffer from high variance and their structure is **unstable** – using different training sets typically gives rise to wildly varying trees.

As an extreme example, simply modifying the level of only one of the predictors in only two observations can yield a tree with a completely different topology, as in Figure 21.10.

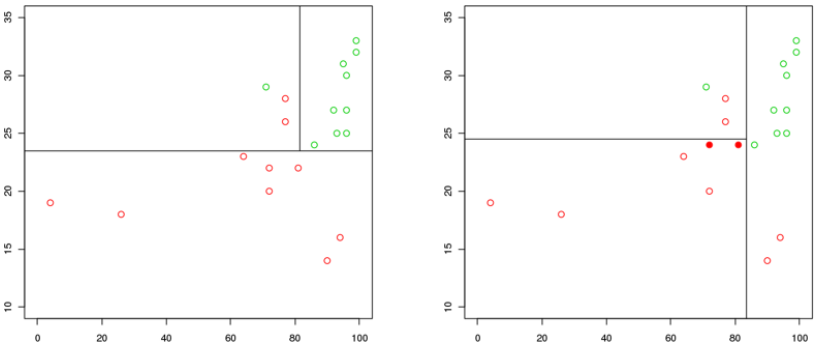
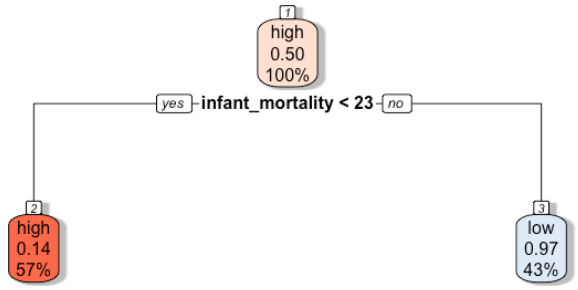


Figure 21.10: Different tree topologies with small changes in the training set (data modified from [19]).

This lack of robustness is a definite strike against the use of CART; despite this, the relative ease of their implementation makes them a popular classification tool.

Examples A classification tree for the response LE in the Gapminder 2011 dataset is shown below:

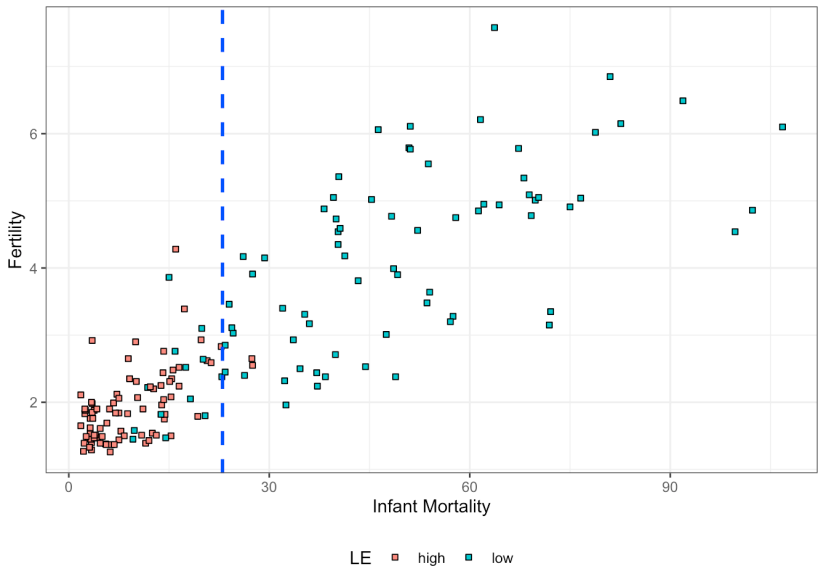
```
reg.tree.2 <- rpart::rpart(LE~fertility +
        infant_mortality + gdp,  data=gapminder.2011)
rpart.plot::rpart.plot(reg.tree.2, box.palette="RdBu",
                       shadow.col="gray", nn=TRUE)
```




```
1) root (166) high/low (0.5 0.5)
  2) infant_mortality< 23 (94) high (0.862 0.138)
  3) infant_mortality>=23 (72) low  (0.028 0.972)
```

The stratification in the scatter plot is shown below.

```
library(ggplot2)
ggplot(gapminder.2011) +
geom_point(aes(fill=LE, x=infant_mortality, y=fertility),
           pch=22) + theme_bw() +
  theme(legend.position = "bottom") +
  geom_vline(xintercept = c(23), linetype="dashed",
  color = "blue", size=1) + xlab("Infant Mortality") +
  ylab("Fertility")
```



Note that this tree should not be used for predictions as it was not built on a training subset of the data.

We now revisit the Iowa Housing Price (VE_Housing.csv, modified from [3]) example of Section 20.5 (*Splines*). We build a CART for the sale price, requiring at least 5 observations per leaf.[30]

We only keep those columns for which there are no missing values, for simplicity's sake; in real-world applications, this is not usually a reasonable strategy (see Chapter 15, *Data Preparation*).

30: Recall that we had built a training set `dat.train` with $n = 1160$ observations relating to the selling price `SalePrice` of houses in Ames, Iowa.

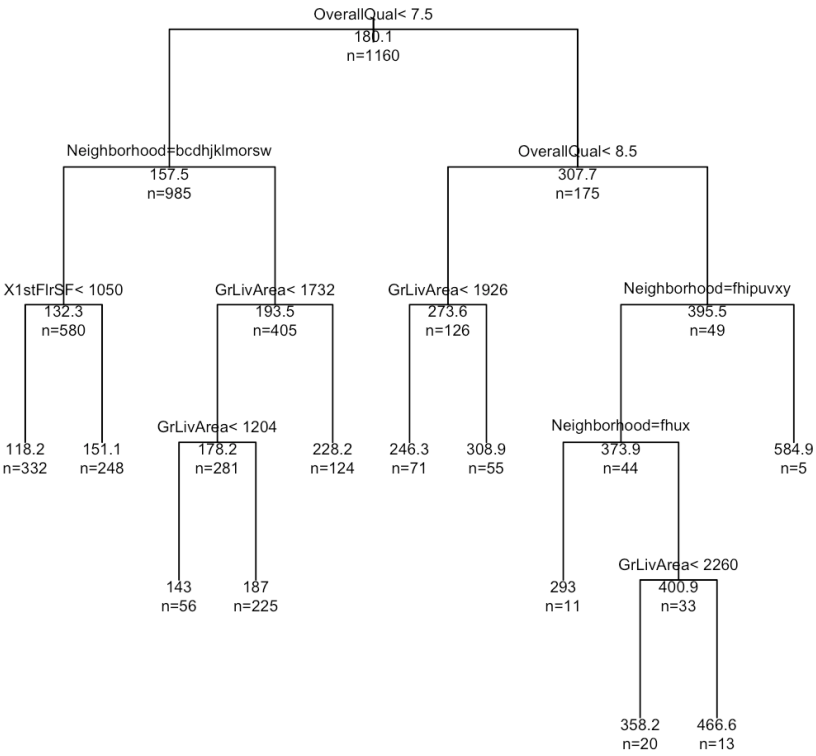
```
n= 1160

node), split, n, deviance, yval
      * denotes terminal node

 1) root 1160 7371073.00 180.1428
   2) OverallQual< 7.5 985 2301952.00 157.4737
     4) Neighborhood=Blueste,BrDale,BrkSide,Edwards,IDOTRR,MeadowV,Mitchel,NAmes,NPkVill,OldTown,
           Sawyer,SWISU 580  672022.40 132.2839
       8) X1stFlrSF< 1050.5 332  240468.90 118.2474 *
       9) X1stFlrSF>=1050.5 248  278574.40 151.0747 *
     5) Neighborhood=Blmngtn,ClearCr,CollgCr,Crawfor,Gilbert,NoRidge,NridgHt,NWAmes,SawyerW,
           Somerst,StoneBr,Timber,Veenker 405  734856.30 193.5480
       10) GrLivArea< 1732.5 281  296100.70 178.2365
         20) GrLivArea< 1204 56   25438.63 143.0286 *
         21) GrLivArea>=1204 225  183967.70 186.9993 *
       11) GrLivArea>=1732.5 124  223588.10 228.2459 *
   3) OverallQual>=7.5 175 1713865.00 307.7376
     6) OverallQual< 8.5 126  498478.90 273.6180
       12) GrLivArea< 1925.5 71  167331.90 246.3000 *
       13) GrLivArea>=1925.5 55  209762.20 308.8831 *
     7) OverallQual>=8.5 49  691521.30 395.4736
       14) Neighborhood=CollgCr,Edwards,Gilbert,NridgHt,Somerst,StoneBr,Timber,Veenker
           44  358089.40 373.9441
         28) Neighborhood=CollgCr,Edwards,Somerst,Timber 11   39962.11 293.0025 *
         29) Neighborhood=Gilbert,NridgHt,StoneBr,Veenker 33  222038.20 400.9246
           58) GrLivArea< 2260 20   42801.90 358.2032 *
           59) GrLivArea>=2260 13   86576.40 466.6498 *
       15) Neighborhood=NoRidge 5  133561.60 584.9336 *
```

```
dat.Housing = read.csv("VE_Housing.csv",
    header=TRUE, stringsAsFactors = TRUE)
missing = attributes(which(apply(is.na(dat.Housing), 2,
    sum) > 0))$names
dat.Housing.new = dat.Housing[,!colnames(dat.Housing) %in%
    missing]
dat.Housing.new = subset(dat.Housing.new, select = -c(Id))

set.seed(1234) # for replicability
n.train = 1160
ind.train = sample(1:nrow(dat.Housing.new), n.train)
dat.train = dat.Housing.new[ind.train,]
dat.test = dat.Housing.new[-ind.train,]
(RT = rpart::rpart(SalePrice ~ ., data=dat.train,
    minbucket=5))
```

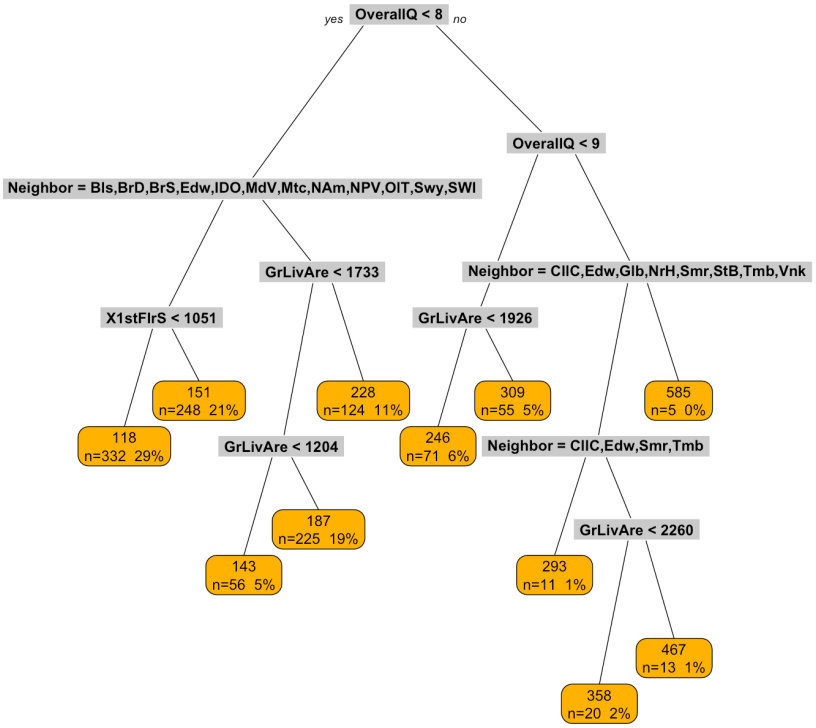
The tree is reasonably complex and fairly difficult to read, especially since there are so many categorical levels in some of the branching nodes.

There are multiple ways to provide a visual display that makes it easier to read the tree.

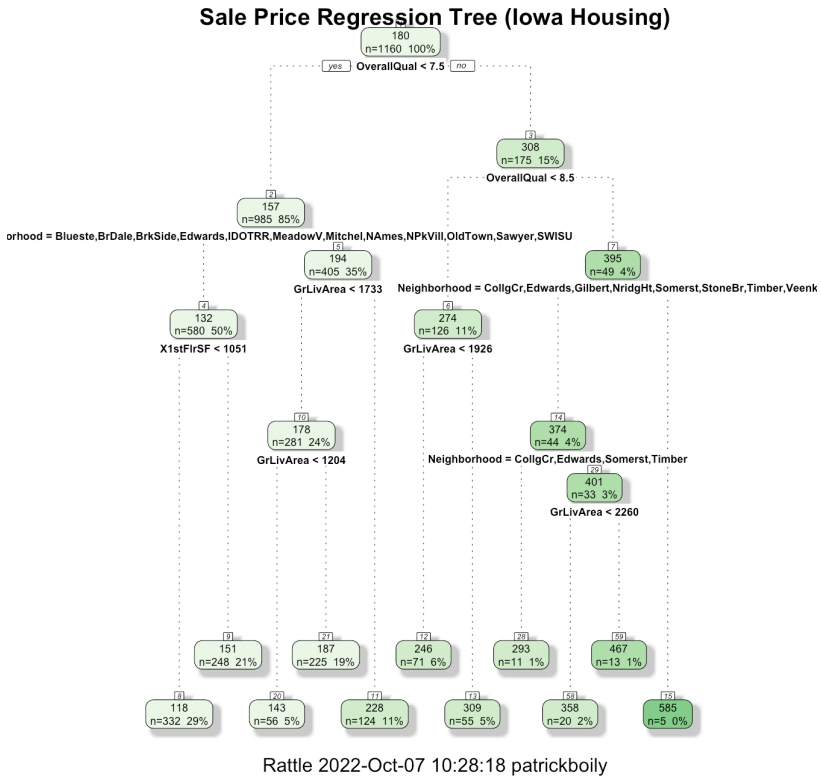
```
plot(RT, margin=0.05, uniform=TRUE)
text(RT, all=TRUE, use.n=TRUE, fancy=FALSE, cex=0.6)
```




```
rpart.plot::prp(RT,extra=101,
    box.col="orange",split.box.col="gray")
```

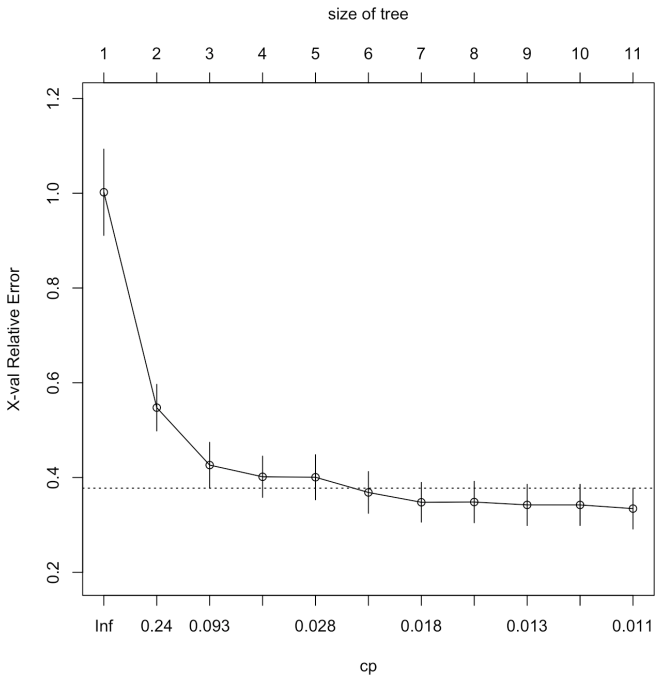



```
rattle::fancyRpartPlot(RT, main="Sale Price Regression Tree
    (Iowa Housing)")
```



These are fully grown trees. Next we use `rpart`'s `plotcp()` to determine how to control the tree's growth (i.e., we prune the tree).

```
set.seed(1234) # for replicability
rpart::plotcp(RT)
```



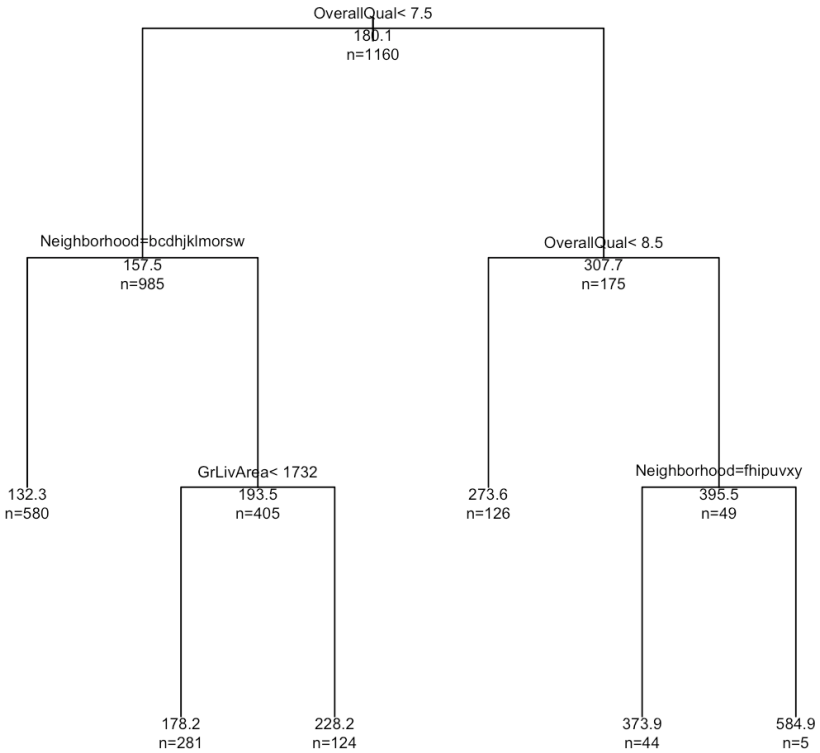
From this plot, we see that the tuning parameter should be around 0.024, so we prune the tree as follows:

```
(RT.p = rpart::prune(RT, cp=0.024))
```

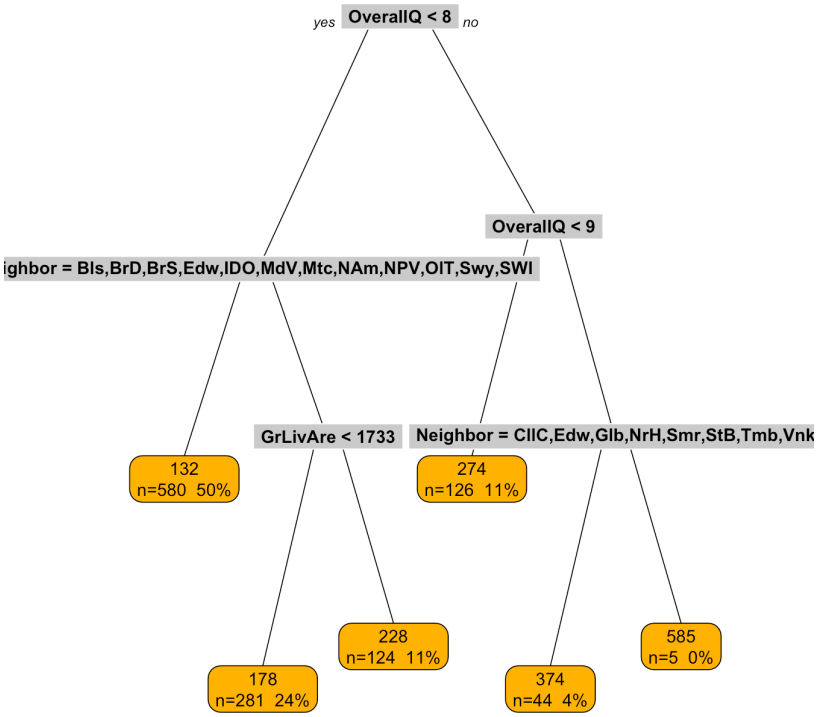
```
n= 1160

node), split, n, deviance, yval
      * denotes terminal node

 1) root 1160 7371073.0 180.1428
   2) OverallQual< 7.5 985 2301952.0 157.4737
     4) Neighborhood=Blueste,BrDale,BrkSide,Edwards,IDOTRR,MeadowV,Mitchel,NAmes,NPkVill,OldTown,
           Sawyer,SWISU 580  672022.4 132.2839 *
     5) Neighborhood=Blmngtn,ClearCr,CollgCr,Crawfor,Gilbert,NoRidge,NridgHt,NWAmes,SawyerW,
        Somerst,StoneBr,Timber,Veenker 405  734856.3 193.5480
      10) GrLivArea< 1732.5 281  296100.7 178.2365 *
      11) GrLivArea>=1732.5 124  223588.1 228.2459 *
   3) OverallQual>=7.5 175 1713865.0 307.7376
     6) OverallQual< 8.5 126  498478.9 273.6180 *
     7) OverallQual>=8.5 49  691521.3 395.4736
      14) Neighborhood=CollgCr,Edwards,Gilbert,NridgHt,Somerst,StoneBr,Timber,Veenker
            44  358089.4 373.9441 *
      15) Neighborhood=NoRidge 5  133561.6 584.9336 *
```

We go from 11 to 6 leaves. The structure of the pruned tree is plotted below.

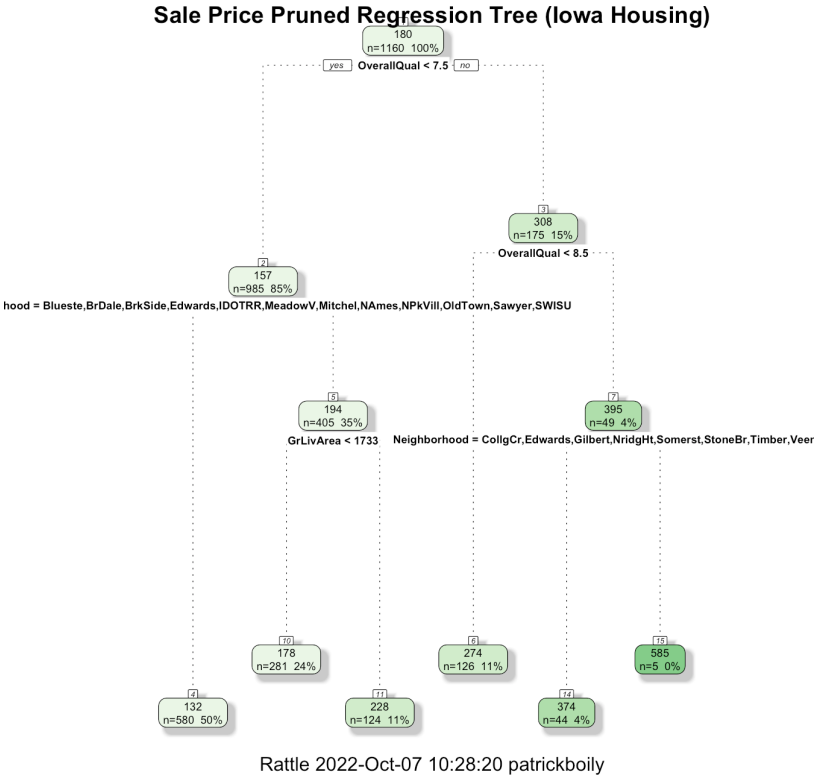
```
plot(RT.p, margin=0.05, uniform=TRUE)
text(RT.p, all=TRUE, use.n=TRUE, fancy=FALSE, cex=0.6)
```



```
rpart.plot::prp(RT.p,extra=101, box.col="orange",
                split.box.col="gray")
```




```
rattle::fancyRpartPlot(RT.p, main="Sale Price Pruned
                       Regression Tree (Iowa Housing)")
```



Now that we have a full regression tree and a pruned tree, our last task is to see how well they perform as predictive models on the test data `dat.test`.

For the full tree, we compute the reduction in SSE using the **predictions**:

```
yhat.RT = predict(RT, dat.test)
SSE.RT = sum((yhat.RT-dat.test$SalePrice)^2)
SSE.average = sum((mean(dat.test$SalePrice) -
                    dat.test$SalePrice)^2)
round((1-SSE.RT/SSE.average), digits=3)
```

```
[1] 0.703
```

For the pruned tree, the corresponding reduction is:

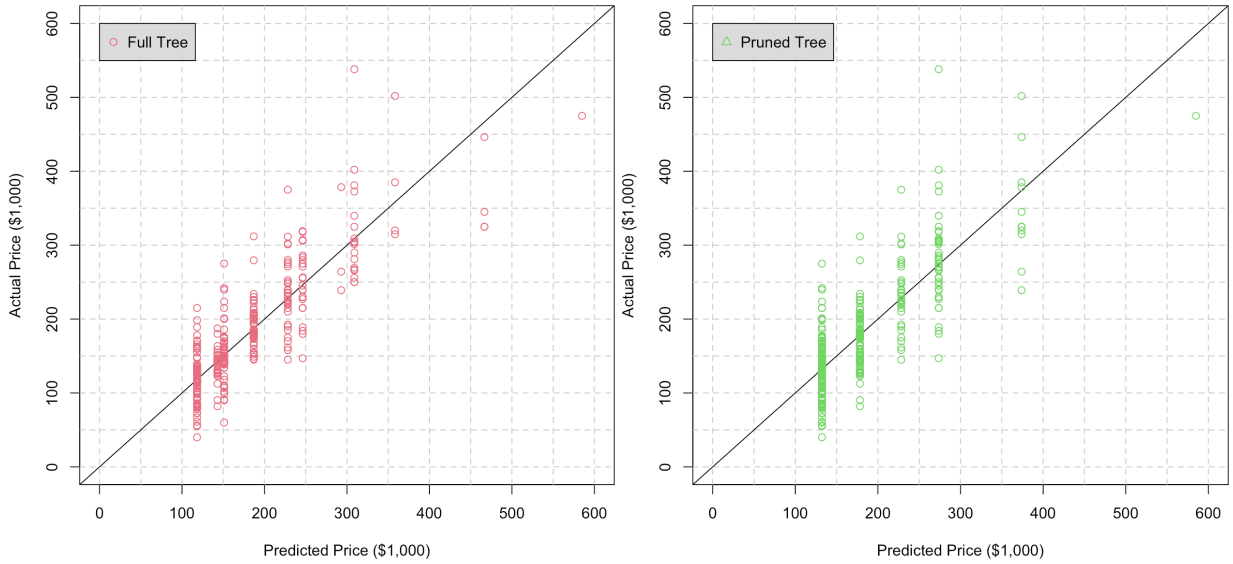
```
yhat.RT.p = predict(RT.p, dat.test)
SSE.RT.p = sum((yhat.RT.p-dat.test$SalePrice)^2)
round((1-SSE.RT.p/SSE.average), digits=3)
```

```
[1] 0.646
```

This suggests that pruning is a reasonable approach, in this case (based on Tr). The predictions of both trees are plotted against the actual `SalePrice` values in the next plots.

```
xlimit = ylimit = c(0,600)

plot(NA, col=2, xlim=xlimit, ylim=ylimit,
     xlab="Predicted Price ($1,000)",
     ylab="Actual Price ($1,000)")
abline(h=seq(0,600, length.out=13), lty=2, col="grey",
       v=seq(0,600, length.out=13))
abline(a=0, b=1)
points(yhat.RT, dat.test$SalePrice, col=2)
legend(0,600, legend=c("Full Tree"), col=c(2),
    pch=rep(1), bg='light grey')

plot(NA, col=2, xlim=xlimit, ylim=ylimit,
     xlab="Predicted Price ($1,000)",
     ylab="Actual Price ($1,000)")
abline(h=seq(0,600, length.out=13), lty=2, col="grey",
       v=seq(0,600, length.out=13))
abline(a=0, b=1)
points(yhat.RT.p, dat.test$SalePrice, col=3)
legend(0,600, legend=c("Pruned Tree"), col=c(3),
    pch=rep(2), bg='light grey')
```

Obviously, there are some departures from the actual response values, but given that the regression trees can only predict a small number of selling prices (corresponding to the tree leaves), these predictions are reasonably accurate.




```
cor(yhat.RT, dat.test$SalePrice)
cor(yhat.RT.p, dat.test$SalePrice)
```

```
[1] 0.8412035
[1] 0.8047534
```

How do these CART predictions compare with the MARS predictions of Section 20.6? The Iowa Housing dataset contains information about sale prices from 2006 to 2010; would you use the model to make predictions about 2022 sale prices?

Classification trees work in the same manner, although the evaluation step can be conducted in two ways: we can build trees that predict class membership (`type="class"`) or probability of class membership (`type="prob"`). Examples of how to work with these `predict()` options are provided in Section 19.7, *Classification: Kyphosis Dataset*.

21.4.2 Support Vector Machines

This next classifier is more sophisticated, from a mathematical perspective. It was invented by computer scientists in the 1990s.

Support vector machines (SVM) attempt to find hyperplanes that separate the classes in the feature space. On the left in Figure 21.11, we see an artificial data with 3 features: X_1 and X_2 (numerical), Y (categorical, represented by different symbols).

We grow a classification tree (perhaps the one shown on the right in Figure 21.11): two of the leaves are pure, but the risk of misclassification is fairly large in the other 2 (at least for that tree).[31] Without access to more features, that tree is as good as it gets.[32]

31: The tree is not unique, obviously, but any other tree with separators parallel to the axes will only be marginally better, at best.

32: To be sure, we could create an intricate decision tree with more than $2^2 = 4$ separating lines, but that is undesirable for a well-fitted tree.

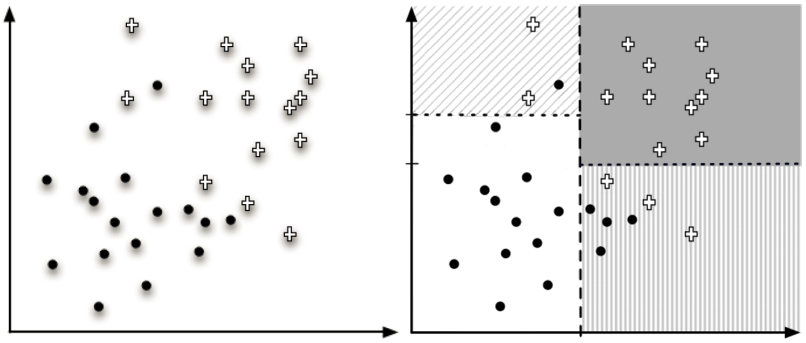


Figure 21.11: Two-class artificial dataset (left) and classification tree (right) [22].

But it is easy to draw a **decision curve** which improves on the effectiveness of the decision tree (see Figure 21.12): a single observation is misclassified by this rule.[33]

33: Perfect separation could lead to overfitting.

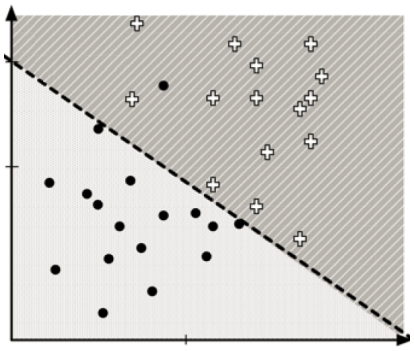


Figure 21.12: Separating hyperplane on a two-class artificial dataset [22].

Separating hyperplanes do not always exist; we may need to:

- extend our notion of **separability**, and/or
- extend the feature space so separation becomes possible.

A **hyperplane** $H_{\boldsymbol{\beta},\beta_0} \subseteq \mathbb{R}^p$ is an affine ("flat") subset of $\mathbb{R}^p$, with

$$\dim\left(H_{\boldsymbol{\beta},\beta_0}\right) = p - 1;$$

in other words, it can be described by

$$H_{\boldsymbol{\beta},\beta_0} : \beta_0 + \boldsymbol{\beta}^\top \mathbf{x} = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p = 0.$$

The vector $\boldsymbol{\beta}$ is normal to $H_{\boldsymbol{\beta},\beta_0}$; if $\beta_0 = 0$, $H_{\boldsymbol{\beta},\beta_0}$ goes through the origin in $\mathbb{R}^p$. Set $F(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}^\top \mathbf{x}$; then $F(\mathbf{x}) > 0$ for points on one "side" of $H_{\boldsymbol{\beta},\beta_0}$ and $F(\mathbf{x}) < 0$ for points on the other.[34]

34: $F(\mathbf{x}) = 0$ for points on $H_{\boldsymbol{\beta},\beta_0}$.

In a binary classification problem with $\mathcal{C} = \{C_1, C_2\} = \{\pm 1\}$, if

$$y_i F(\mathbf{x}_i) > 0, \quad \text{for all } (\mathbf{x}_i, y_i) \in \text{Tr}$$

(or, $y_i F(\mathbf{x}_i) < 0$ for all $(\mathbf{x}_i, y_i) \in \text{Tr}$), then $F(\mathbf{x}) = 0$ determines a **separating hyperplane** for Tr (which does not need to be unique, see Figure 21.12), and we say that Tr is **linearly separable**.

Among all separating hyperplanes, the one which provides the widest separation between the two classes is the **maximal margin hyperplane** (MMH); training observations on the boundary of the **separating strip** are called the **support vectors** (see observations in Figure 21.13).

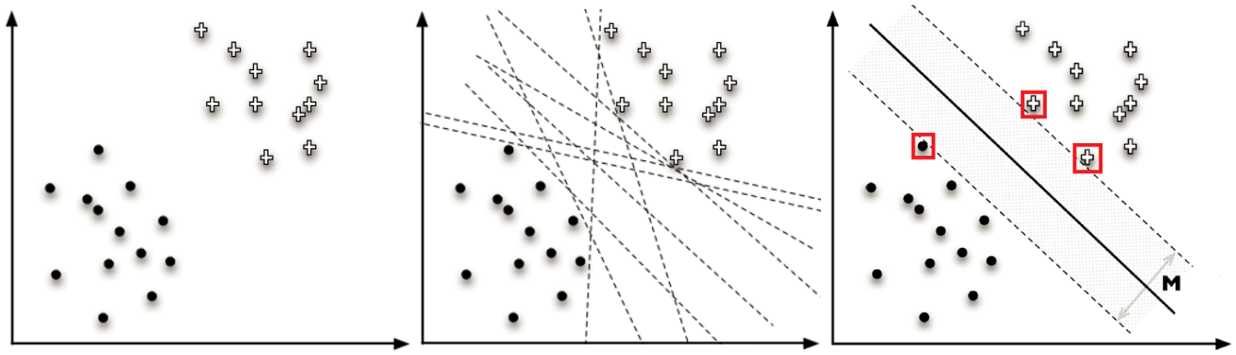


Figure 21.13: Artificial linearly separable subset of a two-class dataset (left), with separating hyperplanes (centre), maximal margin hyperplane with support vectors (right) [22].

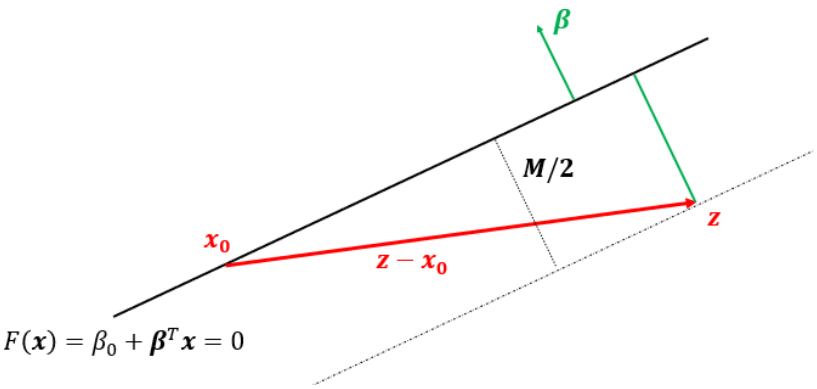
The classification problem simplifies, as always, to a constrained optimization problem:

$$(\boldsymbol{\beta}^*, \beta_0^*) = \arg\max_{(\boldsymbol{\beta},\beta_0)} \{M_{(\boldsymbol{\beta},\beta_0)}\} \quad \text{s.t.} \quad y_i(\beta_0 + \boldsymbol{\beta}\mathbf{x}_i) \geq M_{(\boldsymbol{\beta},\beta_0)}$$

for all $(\mathbf{x}_i, y_i) \in \text{Tr}$, with MMH given by $F(\mathbf{x}) = \beta_0^* + \boldsymbol{\beta}^*\mathbf{x} = 0$.

Any hyperplane can be expressed in an uncountable number of ways; the MMH for which $|F(\mathbf{x}^*)| = 1$ for all support vectors $\mathbf{x}$ provides a **canonical representation**). From geometry, we know that the distance from the canonical maximal margin hyperplane $H_{\boldsymbol{\beta},\beta_0}$ to any point $\mathbf{z}$ can be computed using vector projections.

Let $\mathbf{x}_0$ be a point on MMH, i.e., $F(\mathbf{x}_0) = \beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_0 = 0$, as shown below:



In particular, note that $\beta_0 = -\boldsymbol{\beta}^\top \mathbf{x}_0$. Then,

$$\frac{M}{2} = \text{dist}\left(\mathbf{z}, H_{\boldsymbol{\beta},\beta_0}\right) = \left\|\text{proj}_{\boldsymbol{\beta}}(\mathbf{z} - \mathbf{x}_0)\right\| = \left\|\frac{\boldsymbol{\beta}^\top(\mathbf{z} - \mathbf{x}_0)}{\|\boldsymbol{\beta}\|^2}\boldsymbol{\beta}\right\|$$
$$= \frac{|\boldsymbol{\beta}^\top(\mathbf{z} - \mathbf{x}_0)|}{\|\boldsymbol{\beta}\|^2}\|\boldsymbol{\beta}\| = \frac{|\boldsymbol{\beta}^\top\mathbf{z} - \boldsymbol{\beta}^\top\mathbf{x}_0|}{\|\boldsymbol{\beta}\|} = \frac{|F(\mathbf{z})|}{\|\boldsymbol{\beta}\|}.$$

If $\mathbf{z}$ is a support vector, then $|F(\mathbf{z})| = 1$, and

$$\frac{M}{2} = \text{dist}\left(\mathbf{z}, H_{\boldsymbol{\beta},\beta_0}\right) = \frac{1}{\|\boldsymbol{\beta}\|}.$$

Maximizing the margin M is thus equivalent to minimizing $\frac{\|\boldsymbol{\beta}\|}{2}$, and, since the square function is monotonic,

$$\arg\max_{(\boldsymbol{\beta},\beta_0)}\{M \mid y_i(\beta_0 + \beta^\top \mathbf{x}_i) \geq 1,\ \forall \mathbf{x}_i \in \text{Tr}\}$$

is equivalent to

$$\arg\min_{(\boldsymbol{\beta},\beta_0)}\left\{\frac{1}{2}\|\boldsymbol{\beta}\|^2 \;\middle|\; y_i(\beta_0 + \beta^\top \mathbf{x}_i) \geq 1,\ \forall \mathbf{x}_i \in \text{Tr}\right\}.$$

This constrained quadratic problem (QP) can be solved by Lagrange multipliers (in implementations, it is solved numerically), but a key observation is that it is possible to rewrite

$$\boldsymbol{\beta} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i, \quad \text{with} \quad \sum_{i=1}^{N} \alpha_i y_i = 0$$

thanks to the **representer theorem**.[35]

35: Technically speaking we do not need to invoke the representer theorem in the linear separable case. At any rate, the result is out-of-scope for this document.

The original QP becomes

$$\arg\min_{(\boldsymbol{\beta},\beta_0)}\left\{\frac{1}{2}\sum_{i,j=1}^{N} \alpha_i \alpha_j \mathbf{x}_i^\top \mathbf{x}_j - \sum_{i=1}^{N} \alpha_i \;\middle|\; \sum_{i=1}^{N} \alpha_i y_i = 0,\ \forall \mathbf{x}_i, \mathbf{x}_j \in \text{Tr}\right\}.$$

Ultimately, it can be shown that all but L of the coefficients α_i are 0, typically, $L \ll N$.[36] The **decision function** is defined by

36: The **support vectors** are those training observations $\mathbf{x}_{i_k}$, $k = 1, \ldots, L$, for which $\alpha_{i_k} \neq 0$.

$$T(\mathbf{x}; \boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i + \beta_0 = \sum_{k=1}^{L} \alpha_{i_k} y_{i_k} \mathbf{x}_{i_k}^\top \mathbf{x} + \beta_0,$$

scaled so that $T(\mathbf{x}_{i_k}; \alpha) = y_{i_k} = \pm 1$ for each support vector $\mathbf{x}_{i_k}$.

The **class assignment** for any $\mathbf{x} \in \text{Te}$ is thus

$$\text{class}(\mathbf{x}) = \begin{cases} +1 & \text{if } T(\mathbf{x}; \alpha) \geq 0 \\ -1 & \text{if } T(\mathbf{x}; \alpha) < 0 \end{cases}$$

In practice (especially when $N < p$), the data is rarely linearly separable into distinct classes (as below, for instance).

Additionally, even when the classes are linearly separable, the data may be **noisy**, which could lead to overfitting, with technically optimal but practically sub-optimal maximal margin solutions (see [22] for examples).

In applications, **support vector classifiers** optimize instead a **soft margin**, one for which some misclassifications are permitted (as in Figure 21.15).

The soft margin problem can be written as

$$\arg\min_{(\boldsymbol{\beta},\beta_0)}\left\{\frac{1}{2}\boldsymbol{\beta}^\top\boldsymbol{\beta} \;\middle|\; y_i(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i) \geq 1 - \varepsilon_i,\ \varepsilon_i \geq 0, \forall \mathbf{x}_i \in \text{Tr},\ \|\boldsymbol{\varepsilon}\| < C\right\},$$

where C is a (budget) **tuning parameter**, $\boldsymbol{\varepsilon}$ is a vector of slack variables, canonically scaled so that $|F(\mathbf{x}^*)| = |\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}^*| = 1$ for any eventual support vector $\mathbf{x}^*$.

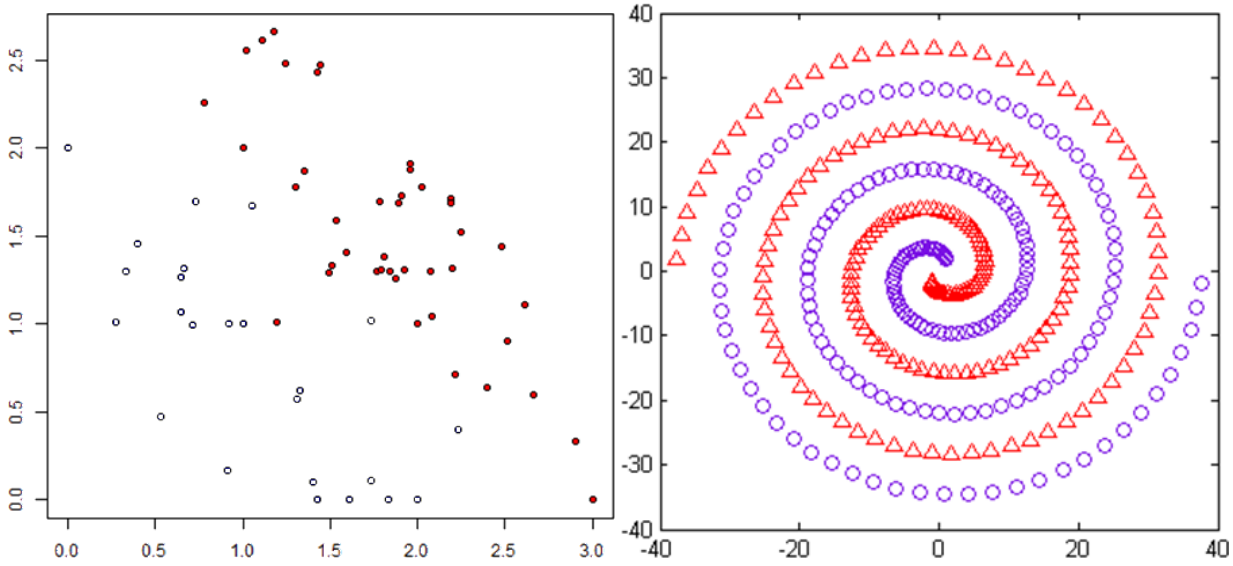


Figure 21.14: Non-linearly separable two-class datasets.

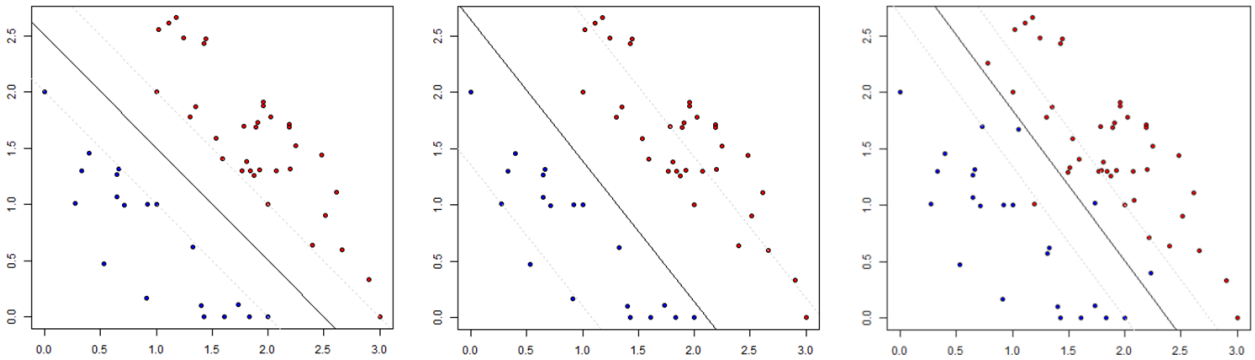


Figure 21.15: Hard margin for a linearly separable classifier (left); soft margin for a linearly separable classifier (middle); soft margin for a non-linearly separable classifier (right).

Such a model offers greater robustness against unusual observations, while still classifying most training observations correctly:

- if $\varepsilon_i = 0$, then $\mathbf{x}_i \in \text{Tr}$ is **correctly classified**;[37]
- if $0 < \varepsilon_i < 1$, then $\mathbf{x}_i \in \text{Tr}$ is **acceptably classified**;[38]
- if $\varepsilon_i \geq 1$, it is **incorrectly classified**.

37: It falls on the correct side of the hyperplane, and outside the maximum margin.

38: It falls on the correct side of the hyperplane, but within the margin.

If $C = 0$, then no violations are allowed ($\|\boldsymbol{\varepsilon}\| = 0$) and the problem reduces to the hard margin SVM classifier; a solution may not even exist if the data is not linearly separable.

If $C > 0$ is an integer, no more than C training observations can be misclassified; indeed, if $i_1, \ldots, i_C$ are the misclassified indices, then $\varepsilon_{i_1}, \ldots, \varepsilon_{i_C} \geq 1$ and

$$C \geq \sum_{i=1}^{N} \varepsilon_i \geq \sum_{k=1}^{C} \varepsilon_{i_k} \geq C.$$

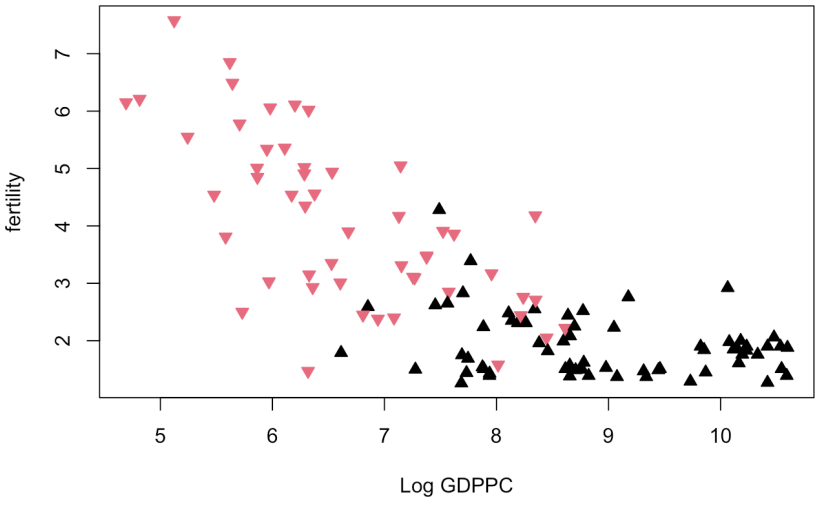
As C increases, tolerance for violations also increases, as does the width of the soft margin; C plays the role of a **regularization parameter**, and is usually selected *via* cross-validation.

Low values of C are associated with harder margins, which leads to low bias but high variance (a small change in the data could create qualitatively different margins); large values of C are associated with wider (softer) margins, leading to more potential misclassifications and higher bias, but also lower variance as small changes in the data are unlikely to change the margin significantly.

We can build a classifier through the representer theorem formulation as before, the only difference being that the **decision function** $T(\mathbf{x}; \boldsymbol{\alpha})$ is scaled so that $|T(\mathbf{x}_{i_k}; \boldsymbol{\alpha})| \geq 1 - \varepsilon_{i_k}$ for every support vector $\mathbf{x}_{i_k}$. It is difficult to determine what the value of the regularization parameter C should be at first glance; an optimal value can be obtained *via* a **tuning process**, which tries out various values and identifies the one that produces an optimal model.

Example We train a SVM with $C = 0.1$ (obtained *via* a tuning procedure for C) for the 2011 Gapminder dataset to predict the life expectancy class Y in terms of the fertility rate X_1 and the logarithm of GDP per capita X_2; $n = 116$ observations are used in the training set `gapminder.2011.tr`, the rest are set aside in the test set `gapminder.2011.te`.

```
set.seed(0)
ind.train = sample(nrow(gapminder.2011),
                   round(0.7*nrow(gapminder.2011)),
                   replace=FALSE)
gapminder.2011.tr = gapminder.2011[ind.train,]
gapminder.2011.te = gapminder.2011[-ind.train,]

x <- gapminder.2011.tr[,c("fertility","gdp","population")]
w <- log(x[,2]/x[,3])
x <- data.frame(x[,1],w)
y <- gapminder.2011.tr[,c("LE")]
dat = data.frame(x,y)
plot(w,x[,1],col=y,bg=y,pch=(as.numeric(y)+23),
     xlab="Log GDPPC", ylab="fertility")
```



The red triangles represent countries with low life expectancy; the black ones, countries with high life expectancy. Notice the class overlap in the training data.

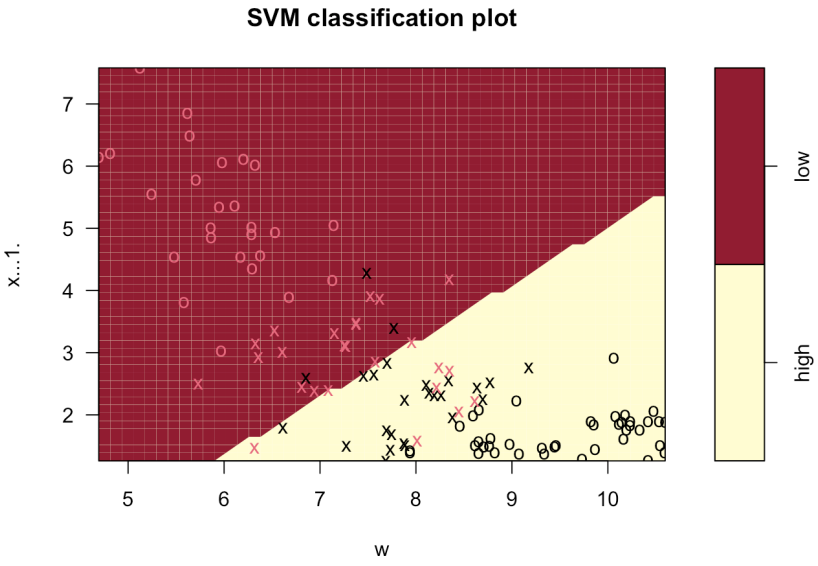
We run 7 linear SVM models with various cost parameters (through `e1071`'s `tune()` function), the optimal model has $C = 0.1$.

```
library(e1071)
tuned.model <- tune(svm, y~., data = dat, kernel = "linear",
        ranges = list(cost = c(0.001, 0.01, 0.1,
                        1, 5, 10, 100)))
(best.mod <- tuned.model$best.model)
```

```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.1

Number of Support Vectors:  50
```

The corresponding SVM model is obtained *via* the `svm()` function (note the parameters). The SVM decision boundary is shown below:

```
svmfit <- svm(y~., data = dat, kernel = "linear", cost=0.1)
plot(svmfit, dat, main="Linear Kernel")
```



We can evaluate the model's performance on Te (`dat.te`); the confusion matrix of the model on the test set is:

```
x <- gapminder.2011.te[,c("fertility","gdp","population")]
w <- log(x[,2]/x[,3])
x <- data.frame(x[,1],w)
y <- gapminder.2011.te[,c("LE")]

# Test data
```

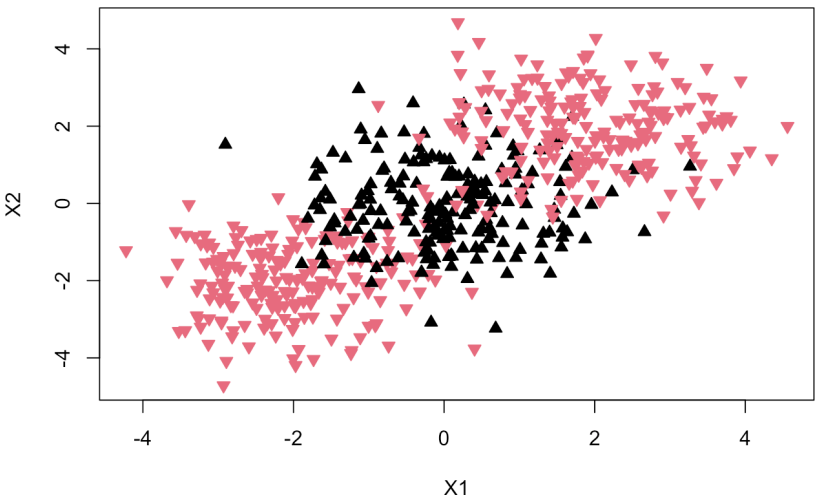
```
dat.te = data.frame(x,y)
# Class prediction on test data
results = predict(svmfit,dat.te)
# Confusion matrix
table(actual=gapminder.2011.te$LE,pred=results)
```

$\alpha = 0.5$		**prediction**	
		0	1
actual	0	22	10
	1	1	17

It is not a perfectly accurate model, but it is certainly acceptable given the class overlap in Tr.

Nonlinear Boundaries If the boundary between two classes is linear, the SVM classifier of the previous section is a natural way to attempt to separate the classes. In practice, however, the classes are rarely so cleanly separated, as below, say.

```
set.seed(0)
x <- matrix(rnorm(600*2), ncol = 2)
y <- c(rep(-1,200), rep(0,200),rep(1,200))
x[y==1,] <- x[y==1,] + 2
x[y==-1] <- x[y==-1,] - 2
y <- y^2
dat <- data.frame(x=x, y=as.factor(y))
plot(dat[,1], dat[,2], col=dat[,3], bg=dat[,3],
     pch=(as.numeric(dat[,3])+23), xlab="X1", ylab="X2")
```



In both the hard and the soft margin support vector classifiers, the function to optimize takes the form

$$\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j\mathbf{x}_i^\top\mathbf{x}_j - \sum_{i=1}^{N}\alpha_i,$$

and the decision function, the form

$$T(\mathbf{x}; \boldsymbol{\alpha}) = \sum_{k=1}^{L} \alpha_{i_k} y_{i_k} \mathbf{x}_{i_k}^\top \mathbf{x} + \beta_0.$$

However, we do not actually need to know the support vectors $\mathbf{x}_{i_k}$ (or even the observations $\mathbf{x}_i$, for that matter) in order to compute the decision function values – it is sufficient to have access to the **inner products** $\mathbf{x}_i^\top \mathbf{x}_j$ or $\mathbf{x}_{i_k}^\top \mathbf{x}$, which are usually denoted by $\langle \mathbf{x}_{i_k}, \mathbf{x} \rangle$ or $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$.

The objective function and the decision function can thus be written as

$$\frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \sum_{i=1}^{N} \alpha_i, \quad T(\mathbf{x}; \boldsymbol{\alpha}) = \sum_{k=1}^{L} \alpha_{i_k} y_{i_k} \langle \mathbf{x}_{i_k}, \mathbf{x} \rangle + \beta_0.$$

This seemingly innocuous remark opens the door to the **kernel approach**; we could conceivably replace the inner products $\langle \mathbf{x}, \mathbf{w} \rangle$ by generalized inner products $K(\mathbf{x}, \mathbf{w})$, which provide a measure of similarity between the observations $\mathbf{x}$ and $\mathbf{w}$.

Formally, a **kernel** is a symmetric (semi-)positive definite operator $K : \mathbb{R}^p \times \mathbb{R}^p \longrightarrow \mathbb{R}_0^+$.[39] Common statistical learning kernels include:

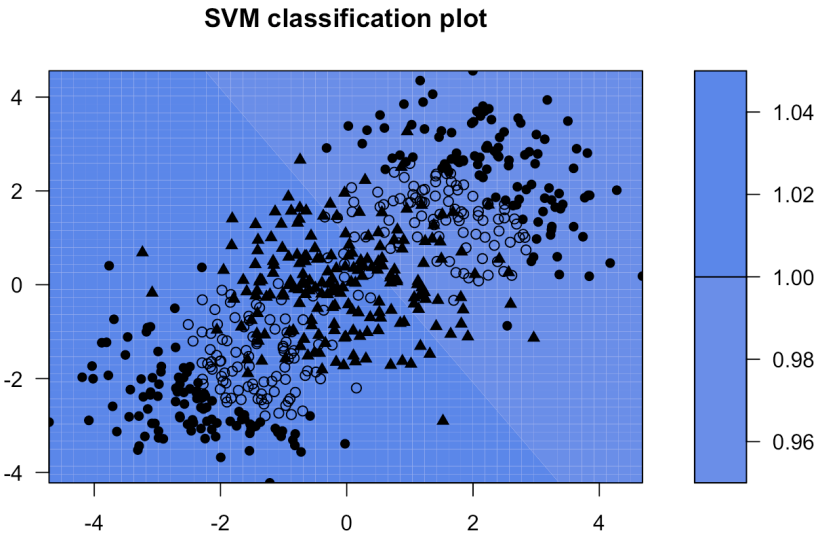
- **linear** – $K(\mathbf{x}, \mathbf{w}) = \mathbf{x}^\top \mathbf{w}$;
- **polynomial of degree** d – $K_d(\mathbf{x}, \mathbf{w}) = (1 + \mathbf{x}^\top \mathbf{w})^d$;
- **Gaussian** (or radial) – $K_\gamma(\mathbf{x}, \mathbf{w}) = \exp(-\gamma \|\mathbf{x} - \mathbf{w}\|_2^2)$, $\gamma > 0$;
- **sigmoid** – $K_{\kappa,\delta}(\mathbf{x}, \mathbf{w}) = \tanh(\kappa \mathbf{x}^\top \mathbf{w} - \delta)$, for allowable κ, δ.

39: By analogy with positive definite square matrices, this means that $\sum_{i,j=1}^{N} c_i c_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0 \; \forall \mathbf{x}_i \in \mathbb{R}^p, c_j \geq 0$.

For instance, a linear kernel SVM and a radial kernel SVM with $\gamma = 1$, $C = 0.5$ yield the following classifications on the previous dataset.[40]

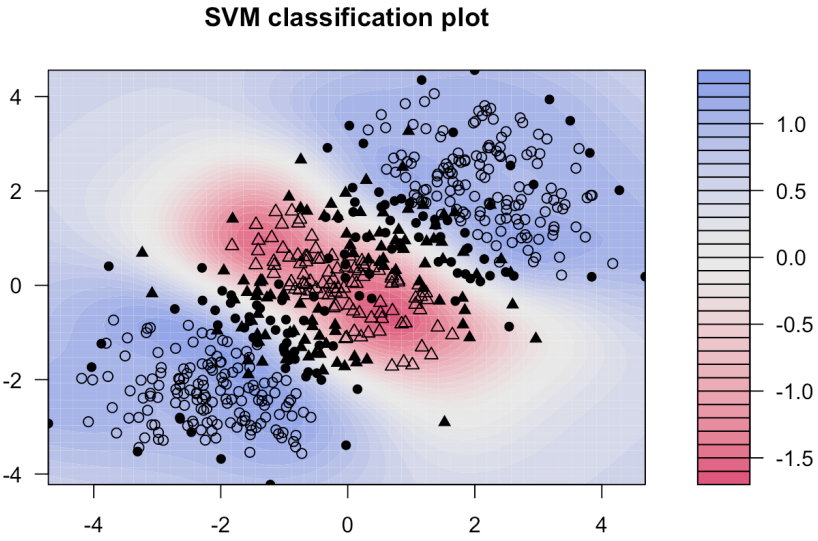
40: We are using `kernlab`'s `ksvm()` function and display the linear SVM output for comparison, whose performance we expect to be crap-tastic

```
library(kernlab)

# linear SVM
kernfit.lin <- ksvm(x,y, type = "C-svc",
                    kernel = 'vanilladot', C = 10)
kernlab::plot(kernfit.lin, data=x)
```



Not that great, to be honest...

```
# Gaussian SVM
kernfit.rbf <- ksvm(x,y, type = "C-svc", kernel = 'rbfdot',
                    sigma=1, C = 0.5)
kernlab::plot(kernfit.rbf, data=x)
```



How is the decision boundary computed? The principle is the same as with linear SVM: the objective function and the decision function are

$$\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j K(\mathbf{x}_i,\mathbf{x}_j)-\sum_{i=1}^{N}\alpha_i, \quad T(\mathbf{x};\boldsymbol{\alpha})=\sum_{k=1}^{L}\alpha_{i_k}y_{i_k}K(\mathbf{x}_{i_k},\mathbf{x})+\beta_0.$$

For the radial kernel, for instance, if a test observation $\mathbf{x}$ is near a training observation $\mathbf{x}_i$, then $\|\mathbf{x}-\mathbf{x}_i\|_2^2$ is small and $K_\gamma(\mathbf{x},\mathbf{x}_i)\approx 1$; if they are far from one another, then $\|\mathbf{x}-\mathbf{x}_i\|_2^2$ is large and $K_\gamma(\mathbf{x},\mathbf{x}_i)\approx 0$.

In other words, in the radial kernel framework, only those observations close to a test observation play a role in class prediction.

Kernel Trick But why even use kernels in the first place? While the linear kernel is easier to interpret and implement, not all data sets are linearly separable, as we have just seen. Consider the toy classification problem on the left of Figure 21.16 (adapted from an unknown online source).

The optimal margin separating "strip" is obviously not linear. One way out of this problem is to introduce a **transformation** Φ from the original $X-$feature space to a higher-dimensional (or at least, of the same dimension) $Z-$feature space in which the data is linearly separable, and to build a linear SVM on the transformed training observations $\mathbf{z}_i=\Phi(\mathbf{x}_i)$.[41]

In this example, we use some $\Phi:\mathbb{R}^2\to\mathbb{R}^3$; the projection of the transformation into the Z_1Z_3-plane could be as in Figure 21.16 (right).

41: This might seem to go against reduction strategies used to counter the curse of dimensionality; the added dimensions are needed to "unfurl" the data, so to speak.

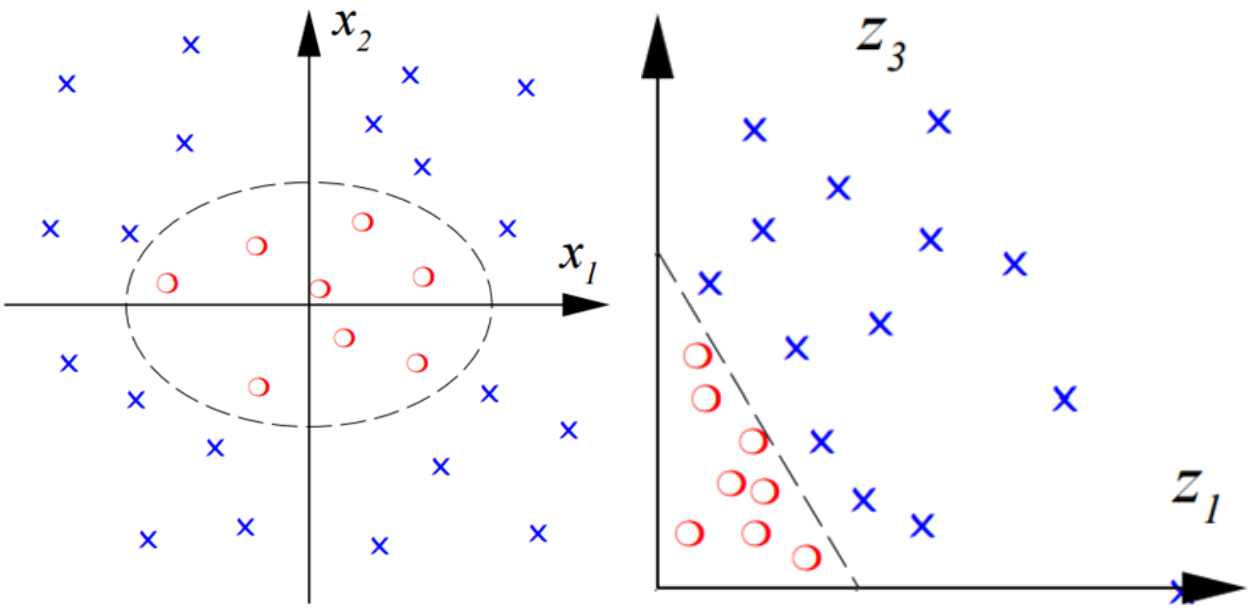


Figure 21.16: Toy classification problem (left); corresponding projection of the linear problem in $Z-$space [author unknown].

The objective function and the decision function take the form

$$\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j\Phi(\mathbf{x}_i)^\top\Phi(\mathbf{x}_j)-\sum_{i=1}^{N}\alpha_i,$$

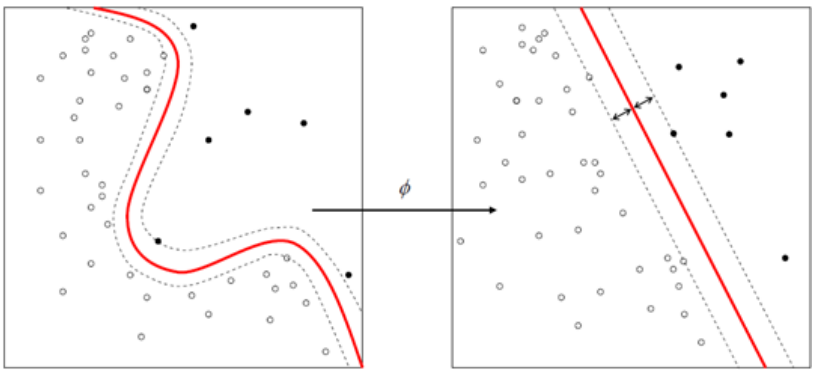
$$T(\mathbf{x};\boldsymbol{\alpha})=\sum_{k=1}^{L}\alpha_{i_k}y_{i_k}\Phi(\mathbf{x}_{i_k})^\top\Phi(\mathbf{x})+\beta_0,$$

and the linear SVM is built as before (but in $Z-$space, not in $X-$space).

It sounds straightforward, but it does take a fair amount of experience to recognize that one way to separate the data is to use

$$\mathbf{z}=\Phi(\mathbf{x})=(x_1^2,\sqrt{2}x_1x_2,x_2^2).$$

And this is one of the easy transformations: what should be used in the case below (image taken from Wikipedia)?



The **kernel trick** simply states that Φ can remain unspecified if we replace $\Phi(\mathbf{x})^\top\Phi(\mathbf{w})$ by a "reasonable" (often radial) kernel $K(\mathbf{x},\mathbf{w})$.

General Classification What do we do if the response variable has $K > 2$ classes? In the **one-versus-all** (OVA) approach, we fit K different 2−class SVM decision functions $T_k(\mathbf{x}; \boldsymbol{\alpha})$, $k = 1, \ldots, K$; in each, one class versus the rest. The test observation $\mathbf{x}^*$ is assigned to the class for which $T_k(\mathbf{x}^*; \boldsymbol{\alpha})$ is largest.

In the **one-versus-one** (OVO) approach, we fit all $\binom{K}{2}$ pairwise 2−class SVM classifiers $\text{class}_{k,\ell}(\mathbf{x})$, for training observations with levels k, ℓ, where $k > \ell = 1, \ldots, K-1$. The test observation $\mathbf{x}^*$ is assigned to the class that wins the most pairwise "competitions".

If K is large, $\binom{K}{2}$ might be too large to make OVO computationally efficient; when it is small enough, OVO is the recommended approach.

Example The `vowel` dataset was taken from the *openML* website. This modified version, by Turney, is based on Robinson's *Deterding Vowel Recognition Data*, which is a speaker-independent recognition of the eleven steady state vowels of British English using a specified training set of `lpc`-derived log area ratios.[42]

We start by reading in the data and summarizing it – the dataset has $n = 990$ observations and $p = 14$ variables.

```
vowel <- read.csv("datasets-uci-vowel.csv", header=TRUE,
                  sep=",", stringsAsFactors=TRUE)
str(vowel)
```

```
'data.frame':   990 obs. of  14 variables:
 $ Train.or.Test: Factor w/ 2 levels "Test","Train": 2 2 2 2 2 2 2 2 2 2 ...
 $ Speaker.Name : Factor w/ 15 levels "Andrew","Bill",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ Speaker.Sex  : Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 2 2 2 2 ...
 $ Feature.0    : num  -3.64 -3.33 -2.12 -2.29 -2.6 ...
 $ Feature.1    : num  0.418 0.496 0.894 1.809 1.938 ...
 $ Feature.2    : num  -0.67 -0.694 -1.576 -1.498 -0.846 ...
 $ Feature.3    : num  1.779 1.365 0.147 1.012 1.062 ...
 $ Feature.4    : num  -0.168 -0.265 -0.707 -1.053 -1.633 ...
 $ Feature.5    : num  1.627 1.933 1.559 1.06 0.764 ...
 $ Feature.6    : num  -0.388 -0.363 -0.579 -0.567 0.394 0.217 0.322 -0.435 -0.512 -0.466 ...
 $ Feature.7    : num  0.529 0.51 0.676 0.235 -0.15 -0.246 0.45 0.992 0.928 0.702 ...
 $ Feature.8    : num  -0.874 -0.621 -0.809 -0.091 0.277 0.238 0.377 0.575 -0.167 0.06 ...
 $ Feature.9    : num  -0.814 -0.488 -0.049 -0.795 -0.396 -0.365 -0.366 -0.301 -0.434 -0.836 ...
 $ Class        : Factor w/ 11 levels "had","hAd","hed",..: 5 6 4 2 11 1 8 7 10 9 ...
```

There is some imbalance in the training/testing set-up (especially as it relates to the speaker sex):

```
table(vowel$Train.or.Test, vowel$Speaker.Sex)
```

```
      Female Male
Test     198  264
Train    264  264
```

42: **Real talk:** we don't actually know what any of that means. But does it matter? Yes, any conclusion we can draw from this dataset will need to be scrutinized by subject matter experts before we can hope to apply them to real-world situations. On the other hand, data is simply marks on paper (or perhaps electromagnetic patterns on the cloud). We can analyze the data without really knowing what the underlying meaning is. The latter approach is usually sterile, but we can always use it to illustrate basic concepts.

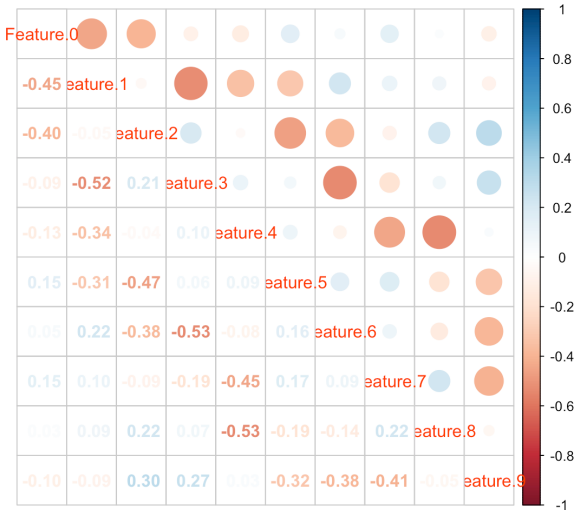
All-in-all, the numerical features seem to be generated from a multivariate normal distribution, with mean vector:

```
colMeans(vowel[,c(4:13)], dims = 1)
```

```
   Feature.0    Feature.1    Feature.2    Feature.3    Feature.4
-3.203740404  1.881763636 -0.507769697  0.515482828 -0.305657576
   Feature.5    Feature.6    Feature.7    Feature.8    Feature.9
 0.630244444 -0.004364646  0.336552525 -0.302975758 -0.071339394
```

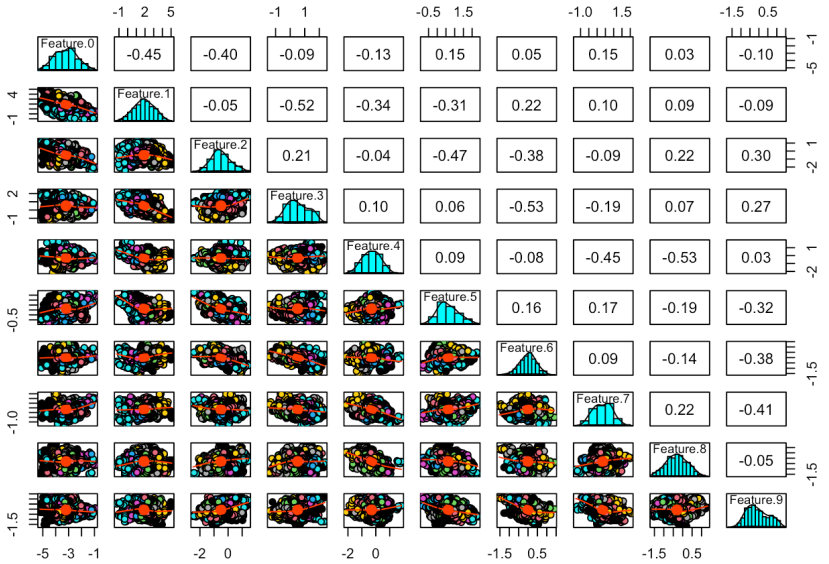
and correlation matrix:

```
corrplot::corrplot.mixed(cor(vowel[,c(4:13)]))
```



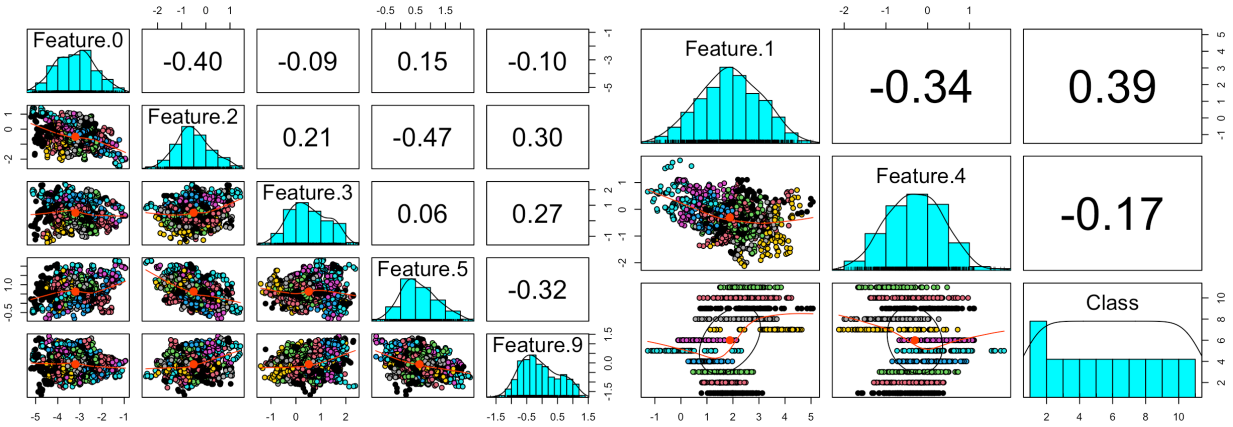
Can we get any information from the paired plots?

```
psych::pairs.panels(vowel[,4:13], pch = 21, bg = vowel$Class)
```



Perhaps if we focus only on certain variables?

```
library(psych)
pairs.panels(vowel[,c(4,6,7,9,13)], pch = 21, bg = vowel$Class)
pairs.panels(vowel[,c(5,8,14)], pch = 21, bg = vowel$Class)
```



The response variable is the `Class`, with 11 levels, and the $p = 10$ predictors are `Feature.0`, ..., `Feature.9`. We train an SVM model on a subset of the vowel dataset. In this instance, we use the training/testing split provided with the data (`Train.or.Test`), but any randomly selected split would be appropriate.

```
training = vowel[vowel$Train.or.Test=="Train",4:14]
testing = vowel[vowel$Train.or.Test=="Test",4:14]
c(nrow(training),nrow(testing))   # training/testing split
```

```
[1] 528 462
```

We use the support vector machine implementation found in the `R` library `e1071`.

First we tune the hyper-parameters on a subsample of the training data by using the `tune()` function, which selects optimal parameters by carrying out a grid search over the specified parameters (otherwise we might spend a lot of time trying to find a good combination of parameters).

For C-classification with a Gaussian kernel, the parameters are

- C, the cost of constraint violation (which controls the penalty paid by the SVM model for misclassifying a training point), and
- γ, the parameter of the Gaussian kernel (used to handle non-linear classification).

If C is "high", then misclassification is costly, and *vice-versa*. If γ is "high", than the Gaussian bump around the points are narrow, and *vice-versa*. Let us run a grid search with C varying from 0.1 to 100 by powers of 10, and $\gamma = 0.5, 1, 2$.

```
vowel.svm.tune.1 <- e1071::tune(e1071::svm,
            train.x=training[,1:10],
            train.y=training[,11],
            kernel="radial",
            ranges=list(cost=10^(-1:2), gamma=c(.5,1,2)))
print(vowel.svm.tune.1)
```

```
Parameter tuning of 'e1071::svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
   10   0.5

- best performance: 0.007619739
```

The minimal misclassfication error (best performance) in this run is reached when the best parameters have the values listed in the output above. Obviously, that search was fairly coarse: searching at a finer level can be very demanding, time-wise.

For comparison's sake, let us see if tuning with finer intervals and larger ranges gives substantially different results.

```
vowel.svm.tune.2 <- e1071::tune(e1071::svm,
            train.x=training[,1:10],
            train.y=training[,11],
            kernel="radial",
            ranges=list(cost=10^(-2:2), gamma=1:20*0.1))
print(vowel.svm.tune.2)
```

```
Parameter tuning of 'e1071::svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
   10   0.8

- best performance: 0.003773585
```

The optimal parameters are sensibly the same, so we might as well stick with the optimal parameters values from the first tuning. Training the model with these values yields:

```
vowel.svm.model = e1071::svm(training[,11] ~ ., data = training,
             type="C-classification",
             cost=10, kernel="radial", gamma=0.5)
summary(vowel.svm.model)
```

```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10

Number of Support Vectors:  351

 ( 27 32 32 26 30 36 37 29 40 32 30 )


Number of Classes:  11

Levels:
 had hAd hed hEd hid hId hod hOd hud hUd hYd
```

Note the number of support vectors. How accurately can this model predict the class of new observations?

```
predicted = predict(vowel.svm.model, testing)
(confusion.matrix = table(pred = predicted, true = testing[,11]))
e1071::classAgreement(confusion.matrix,match.names=TRUE)
```

```
     true
pred  had hAd hed hEd hid hId hod hOd hud hUd hYd
  had  36   0   0   0   0   0   0   0   0   0   0
  hAd   0  40   0   0   0   0   0   0   0   0   0
  hed   0   0  42   0   0   0   0   0   0   0   0
  hEd   0   0   0  37   0   0   0   0   0   0   0
  hid   0   0   0   0  39   0   0   0   0   0   0
  hId   0   0   0   0   2  42   0   0   1   0   0
  hod   0   0   0   0   0   0  36   0   0   0   0
  hOd   0   0   0   0   0   0   0  35   0   0   0
  hud   0   0   0   0   0   0   0   0  23   0   0
  hUd   6   2   0   5   1   0   6   7  18  42  16
  hYd   0   0   0   0   0   0   0   0   0   0  26
$diag
[1] 0.8614719

$kappa
[1] 0.847619

$rand
[1] 0.9425585

$crand
[1] 0.6798992
```

What do you think?

Final Comments In practice, it is not always obvious whether one should use SVM, logistic regression, linear discriminant analysis (LDA), decision trees, etc:[43]

43: In Section 21.5, we argue that it is usually preferable to train a variety of models, rather than just the one.

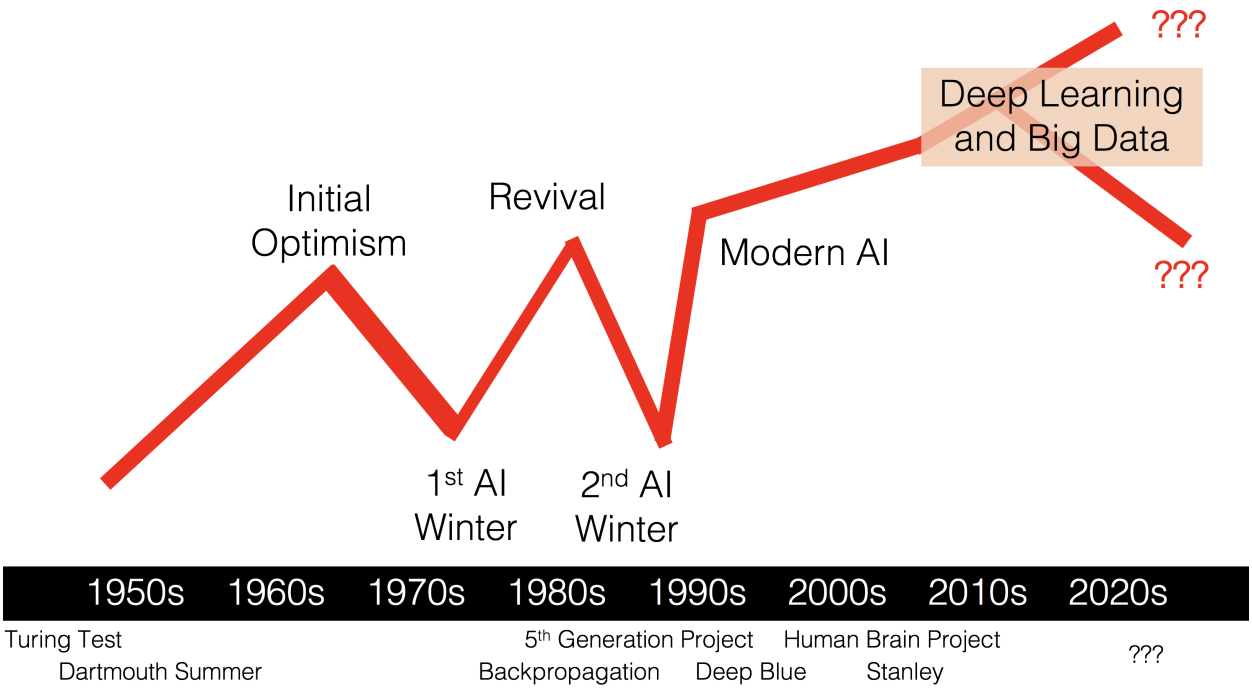


Figure 21.17: Conceptual timeline of the interest and optimism regarding AI; important milestones are indicated below the dates.

- if classes are (nearly) separable, SVM and LDA are usually preferable to logistic regression;
- otherwise, using logistic regression together with a ridge penalty (see Section 20.2, *Shrinkage Methods*) is roughly equivalent to using SVM;
- if the aim is to estimate class membership probabilities, it is preferable to use logistic regression as SVM is not **calibrated**;[44]
- it is possible to use kernels in the logistic regression and LDA frameworks, but at the cost of increased computational complexity.

44: The actual values of $T(\mathbf{x}; \boldsymbol{\alpha})$ have no intrinsic meaning, other than their relative ordering.

All in all, it remains crucial to understand that the *No Free Lunch Theorem* remains in effect [31, 29, 30]. There is simply no magical recipe... although the next technique we discuss is often viewed (and used) as one.

21.4.3 Artificial Neural Networks

When practitioners discuss using **Artificial Intelligence** (AI) techniques [23] to solve a problem, the implicit assumption is often (but not always) that a neural network (or some other variant of **deep learning**) will be used, and for good reason: "neural networks blow all previous techniques out of the water in terms of performance" [11]. But there are some skeletons in the closet: "[...] given the existence of adversarial examples, it shows we really don't understand what's going on" [11].

At various times since Turing's seminal 1950 paper (in which he proposed the celebrated **Imitation Game** [26]), complete artificial intelligence has been announced to be "just around the corner" (see Figure 21.17).

With the advent of **deep learning** and Big Data processing, optimism is as high as it's ever been, but opinions on the topic are varied – to some

commentators, AI is a brilliant success, while to others it is a spectacular failure (see the headlines in Section 14.1.3). So what is really going on?

It is far from trivial to identify the **essential qualities and skills of an intelligence**. There have been multiple attempts to solve the problem by building on Turing's original effort. An early argument by Hofstadter [16] is that any intelligence should:

- provide flexible responses in various scenarios;
- take advantage of lucky circumstances;
- make sense out of contradictory messages;
- recognize the relative importance of a situation's elements;
- find similarities between different situations;
- draw distinctions between similar situations, and
- come up with new ideas from scratch or by re-arranging previous known concepts.

This is not quite the approach taken by modern AI researchers, which define the discipline as the study of **intelligent agents** – any device that perceives its environment and takes actions to maximize its chance of success at some task/goal [27].

Examples include:

- **expert systems** – TurboTax, WebMD, technical support, insurance claim processing, air traffic control, etc.;
- **decision-making** – Deep Blue, auto-pilot systems, "smart" meters, etc.;
- **natural Language Processing** – machine translation, Siri, named-entity recognition, chatGPT, etc.;
- **recommenders** – Google, Expedia, Facebook, LinkedIn, Netflix, Amazon, etc.;
- **content generators** – music composer, novel writer, animation creator, etc.;
- **classifiers** – facial recognition, object identification, fraud detection, etc.

A trained **artificial neural network** (ANN) is a function that maps inputs to outputs in a useful way: it uses a Swiss-army-knife approach to providing outputs – plenty of options are available in the **architecture**, but it's not always clear which ones should be used.

One of the reasons that ANNs are so popular is that the user does not need to decide much about the function or know much about the problem space in advance – ANNs are **quiet models**.

Algorithms allow ANNs to **learn** (i.e. to generate the function and its internal values) automatically; technically, the only requirement is the user's ability to minimize a cost function (which is to say, to be able to solve optimization problems).

Overview The simplest definition of an **artificial neural network** is provided by the inventor of one of the first neuro-computers, R. Hecht-Nielsen, as:

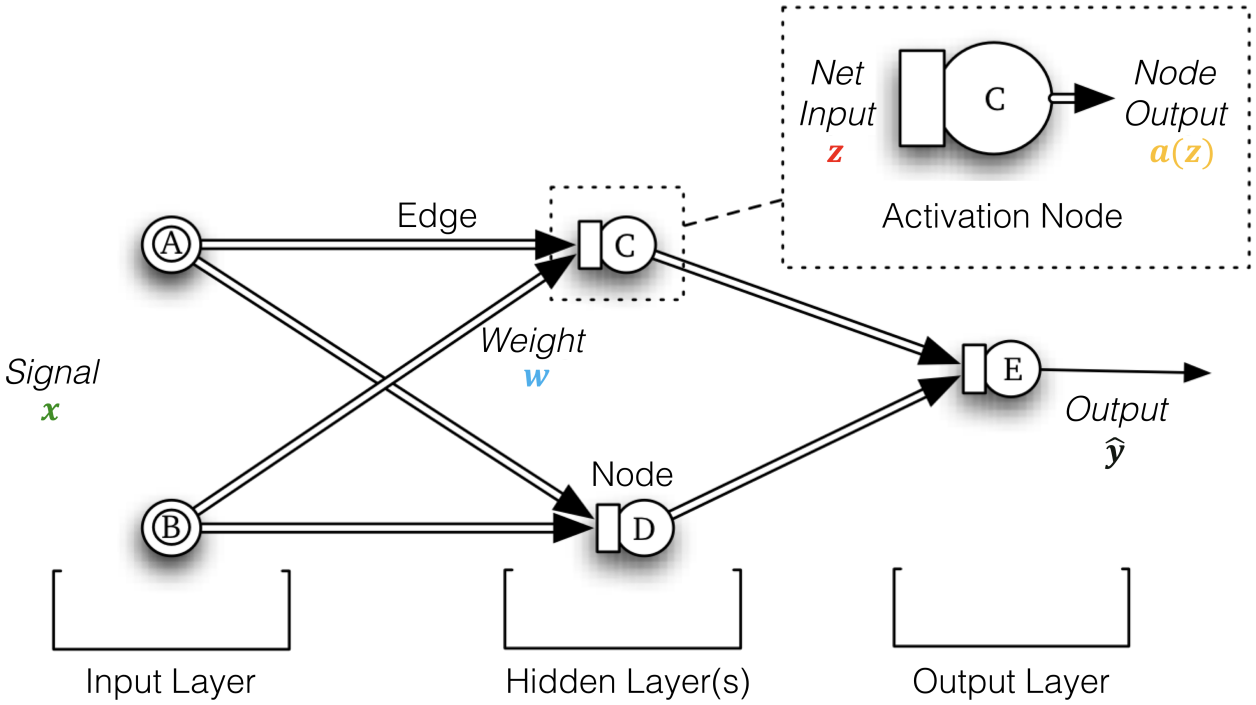


Figure 21.18: Artificial neural network topology – conceptual example. The number of hidden layers is arbitrary, as is the size of the signal and output vectors.

> "[...] a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs. [4]"

An **artificial neural network** is an interconnected group of nodes, inspired by a simplification of neurons in a brain but on much smaller scales. Neural networks are typically organized in **layers**. Layers are made up of a number of interconnected **nodes** which contain an **activation function**.

A **pattern x** (input, signal) is presented to the network *via* the **input layer**, which communicates with one or more **hidden layers**, where the actual processing is done *via* a system of weighted **connections W** (edges).

The hidden layers then link to an **output layer**, which outputs the **predicted response ŷ** (see Figure 21.18).

Neural Networks Architecture In order to train a neural network, we need the following objects [6]:

- some **input data**,
- a number of **layers**,
- a **model**, and
- a **learning process** (loss function and optimizer).

The object interactions is visualized in Figure 21.19.

A network (model), which is composed of layers that are chained together, maps the input data into predictions.[45] The loss function then compares these predictions to the targets, producing a loss value: a measure of how

45: In essence, a neural network is a **function**.

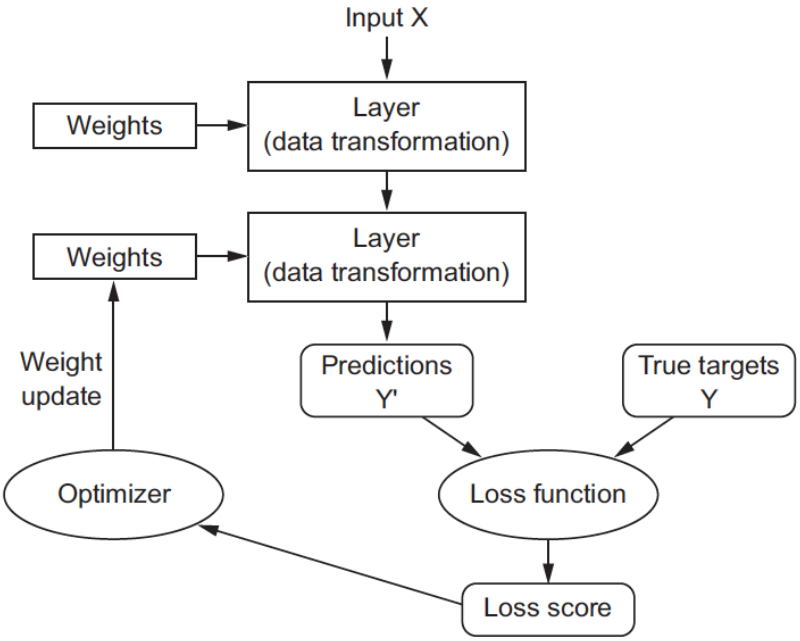


Figure 21.19: Relationship between the network, layers, loss function, and optimizer [6].

well the network's predictions match what was expected. The optimizer uses this loss value to update the network's weights.

Input Data Neural networks start with the **input training data** (and corresponding **targets**) in the form of a **tensor**. Generally speaking, most modern machine learning systems use tensors as their basic data structure. At its core, a tensor is a **container** for data – and it is almost always numerical.

Tensors are defined by three key attributes: their

- **rank** (number of axes) – for instance, a 3D tensor has three axes, while a matrix (2D tensor) has two axes;
- **shape**, a tuple of integers that describes how many dimensions the tensor has along each axis – for instance, a matrix's shape is described using two elements, such as `(3,5)`, a 3D tensor's shape has three elements, such as `(3,5,5)`, a vector (1D tensor)'s shape is given by a single element, such as `(5)`, whereas a scalar has an empty shape, `( )`;
- **data type** – for instance, a tensor's type could be `float32`, `uint8`, `float64`, etc.

Data tensors almost always fall into one of the following categories:

- the most common case is **vector data**; in such datasets, each single data point can be encoded as a vector, and a batch of data will be encoded as a matrix or 2D tensor of shape `(#samples,#features)`, or more simply, as an array of vectors where the first axis is the samples axis and the second axis is the features axis;
- **time series or sequence data**, whenever the passage of time is crucial to the observations in the dataset (or the notion of sequence order), can be stored in a 3D tensor with an explicit time axis; each sample can be encoded as a sequence of vectors (a 2D tensor), and

a batch of data will be encoded as a 3D tensor of shape `(#samples, #timesteps, #features)`, as in Figure 21.20;

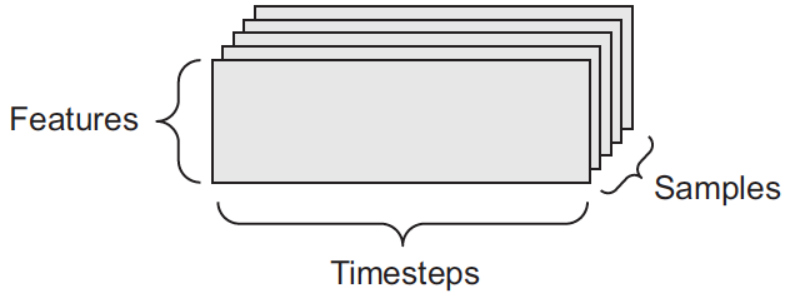


Figure 21.20: A 3D time series data tensor [6].

- **images** typically have three dimensions: height, width, and colour depth;[46] a batch of image data could thus be stored in a 4D tensor of shape `(#samples, #height, #width, #channels)`, as in Figure 21.21;

46: Although grayscale images have only a single colour channel and could thus be stored in 2D tensors, by convention image tensors are always 3D, with a one-dimensional colour channel for grayscale images.

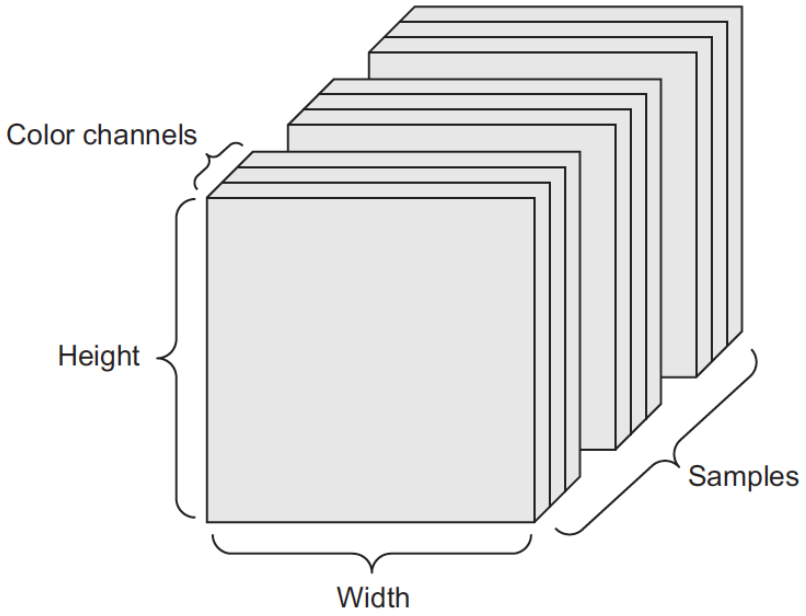


Figure 21.21: A 4D image data tensor [6].

- **video data** is one of the few types of real-world data for which 5D tensors are needed – a video can be understood as a sequence of frames, each frame being a colour image; a sequence of frames can be stored in a 4D tensor `(#frames, #height, #width, #channels)`, and so a batch of different videos can be stored in a 5D tensor of shape `(#samples, #frames, #height, #width, #channels)`.

Layers The core building block of neural networks is the **layer**, a data-processing module that is, in a sense, a **filter** for data: some data goes into the layer and comes out in a more useful form.

Specifically, layers extract **representations** out of the data fed into them – hopefully, representations that are **more meaningful** for the problem at hand. A layer takes as input 1+ tensors and outputs 1+ tensors. Different

layers are appropriate for different tensor formats and different types of data processing.

For instance, simple vector data, stored in 2D tensors, is often processed by **densely connected** layers, also called **fully connected** or **dense** layers. Sequence data, stored in 3D tensors, is typically processed by **recurrent** layers. Image data, stored in 4D tensors, is usually processed by 2D **convolution** layers.

Most of deep learning consists of chaining together simple layers that will implement a form of **progressive data distillation**. However, to build deep learning models in tensor-based modules like Keras [6], it is important to clip together **compatible** layers to form useful data-transformation pipelines.

The notion of **layer compatibility** refers specifically to the fact that every layer can only accept input tensors of a certain shape and return output tensors of a certain shape.

We will discuss tensors in greater detail in Chapter 31.

Model: Networks of Layers An artificial neural network model is essentially a **data processing sieve**, made of a succession of increasingly refined data filters – the **layers**. The most common example of a model is a linear stack of layers, mapping a single input to a single output. Other network topologies include: **two-branch networks**, **multihead networks**, and **inception blocks**. The topology of a network defines a **hypothesis space**.

Since machine learning is basically

> "[...] searching for useful representations of some input data, within a predefined space of possibilities, using guidance from a feedback signal [6],"

by choosing a network topology, we constrain the space of possibilities (hypothesis space) to a specific series of tensor operations, mapping input data to output data.

From a ML perspective, what we are searching for is a good set of values for the weight tensors involved in these tensor operations. Picking the right network architecture is more an art than a science; and although there are some best practices and principles we can rely on, practical experience is the main factor in becoming a proper neural network architect.

Learning Process: Loss Function and Optimizer After a network architecture has been selected, two other objects need to be chosen:

- the **(objective) loss function** is the quantity that is minimized during training – it represents a measure of success for the task at hand, and
- the **optimizer** determines how the network is updated based on the loss function.

In this context, **learning** means finding a combination of model parameters that minimizes the **loss function** for a given set of training data observations and their corresponding targets.

Learning happens by drawing random batches of data samples and their targets, and computing the **gradient** of the network parameters with respect to the **loss** on the batch. The network parameters are then updated by a small amount (the magnitude of the move is defined by the learning rate) in the opposite direction from the gradient.

The entire learning process is made possible by the fact that under a network disguise, neural networks are simply **chains of differentiable tensor operations**, to which it is possible to apply the chain rule of differentiation to find the gradient function mapping the current parameters and current batch of data to a gradient value.

Choosing the right objective function for a given problem is extremely important: the network is ruthless when it comes to lowering its loss function, and it will take any shortcut it can to achieve that objective. If the latter does not fully correlate with success for the task at hand, the network may face unintended side effects.

Simple guidelines exist to help analysts select an appropriate loss function for common problems such as classification, regression, and sequence prediction. We typically use:

- **binary cross entropy** for a binary classification;
- **categorical cross entropy** for a $n-$ary classification;
- **mean squared error** for a regression;
- **connectionist temporal classification** (CTC) for sequence-learning, etc.

In most cases, one of these will do the trick – only when analysts are working on truly new research problems do they have to develop their own objective functions. Let us first illustrate the principles underlying ANNs with a simple example.

We have seen that ANNs are formed from an **input layer** from which the **signal vector x** is inputted, an **output layer** which produces an **output vector $\hat{\mathbf{y}}$**, and any number of **hidden layers**; each layer consists of a number of **nodes** which are connected to the nodes of other layers *via* **directed edges** with associated **weights w**, as seen below.

Nodes from the hidden and output layers are typically **activation nodes** – the output $a(\mathbf{z})$ is some function of the input $\mathbf{z}$. Signals propagate through the ANN using simple arithmetic, once a set of weights $\mathbf{w}$ and activation functions $a(\cdot)$ have been selected (see Figure 21.22).

In a nutshell, at each node, the neural net computes a weighted sum of inputs, applies an activation function, and sends a signal. This is repeated until the various signals reach the final output nodes.

That part is easy – given a signal, an ANN can produce an output, as long as the weights are specified. Matrix notation can simplify the expression for the output $\hat{\mathbf{y}}$ in terms of the signal $\mathbf{x}$, weights $\mathbf{w}$, and activation function $a(\cdot)$.

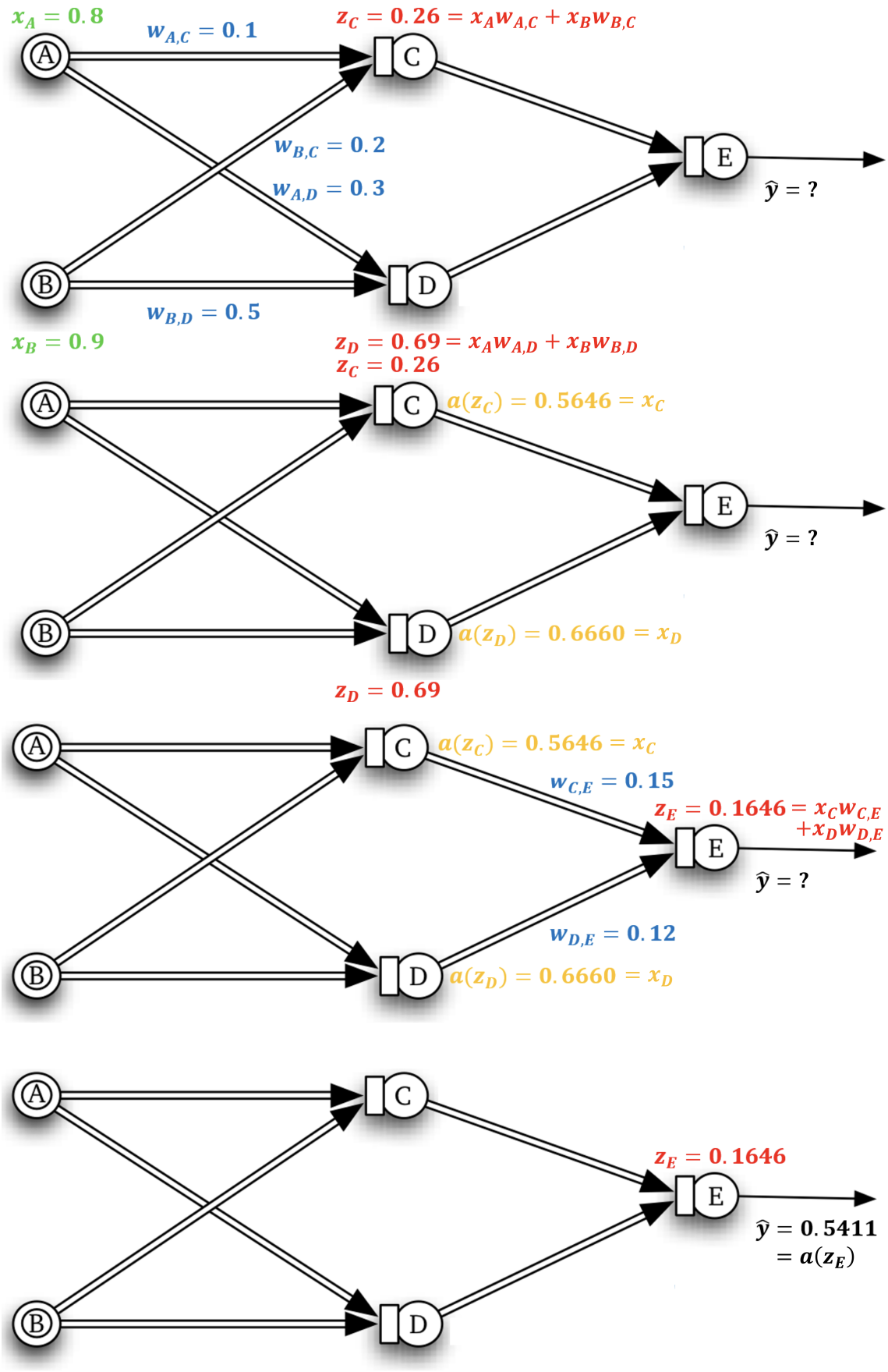


Figure 21.22: Signal propagating forward through an ANN; weights (in blue), activation functions (in yellow), inputs (in green), and output (in black).

For instance, consider the network of Figure 21.22; if

$$a(z) = (1 + \exp(-z))^{-1},$$

the network topology can be re-written as:

- **input layer with p nodes**

$$\mathbf{X}_{N\times p} = \mathbf{X}_{n\times 2} = \begin{bmatrix} x_{A,1} & x_{B,1} \\ \vdots & \vdots \\ x_{A,N} & x_{B,N} \end{bmatrix};$$

- **weights from input layer to hidden layer with M nodes**

$$\mathbf{W}^{(1)}_{p\times M} = \mathbf{W}^{(1)}_{2\times 2} = \begin{bmatrix} w_{AC} & w_{AD} \\ w_{BC} & w_{BD} \end{bmatrix};$$

- **hidden layer with M nodes**

$$\mathbf{Z}^{(2)}_{N\times M} = \mathbf{Z}^{(2)}_{N\times 2} = \begin{bmatrix} z_{C,1} & z_{D,1} \\ \vdots & \vdots \\ z_{C,N} & z_{D,N} \end{bmatrix} = \mathbf{X}\mathbf{W}^{(1)};$$

- **activation function on hidden layer**

$$\mathbf{a}^{(2)} = \begin{bmatrix} (1 + \exp(-z_{C,1}))^{-1} & (1 + \exp(-z_{D,1}))^{-1} \\ \vdots & \vdots \\ (1 + \exp(-z_{C,N}))^{-1} & (1 + \exp(-z_{D,N}))^{-1} \end{bmatrix} = g(\mathbf{Z}^{(2)});$$

- **weights from hidden layer with M nodes to output layer with K nodes**

$$\mathbf{W}^{(2)}_{M\times K} = \mathbf{W}^{(2)}_{2\times 1} = \begin{bmatrix} w_{CE} \\ w_{DE} \end{bmatrix};$$

- **output layer with K nodes**

$$\mathbf{Z}^{(3)}_{N\times K} = \mathbf{Z}^{(3)}_{N\times 1} = \begin{bmatrix} z_{E,1} \\ \vdots \\ z_{E,N} \end{bmatrix} = \mathbf{a}^{(2)}\mathbf{W}^{(2)};$$

- **activation function on output layer**

$$\hat{\mathbf{y}} = \mathbf{a}^{(3)} = \begin{bmatrix} (1 + \exp(-z_{E,1}))^{-1} \\ \vdots \\ (1 + \exp(-z_{E,N}))^{-1} \end{bmatrix} = g(\mathbf{Z}^{(3)});$$

The main problem is that unless the weights are judiciously selected, the output that is produced is unlikely to have anything to do with the desired output. For SL tasks (i.e., when an ANN attempts to emulate the results of training examples), there has to be some method to optimize the choice of the weights against an error function

$$R(\mathbf{W}) = \sum_{i=1}^{N}\sum_{\ell=1}^{k}(\hat{y}_{i,\ell}(\mathbf{W}) - y_{i,\ell})^2 \quad \text{or} \quad R(\mathbf{W}) = -\sum_{i=1}^{N}\sum_{\ell=1}^{k} y_{i,\ell}\ln\hat{y}_{i,\ell}(\mathbf{W})$$

(for value estimation and classification, respectively), where N is the number of observations in the training set, K is the number of output nodes in the ANN, $y_{i,\ell}$ is the known value or class label for the ℓ^{th} output of the i^{th} observation in the training set.

Enter **backpropagation**, which is simply an application of calculus' **chain rule** to $R(\mathbf{W})$. Under reasonable regularity condition, the desired **minimizer** $\mathbf{W}^*$ satisfies $\nabla R(\mathbf{W}^*) = 0$ and is found using **numerical gradient descent**.

Gradient-Based Optimization Initially, the weight matrix $\mathbf{W}$ is filled with small random values (a step called **random initialization**). The weights are then gradually **trained** (or learned), based on a **feedback signal**. This occurs within a **training loop**, which works as follows:

1. draw a batch of training samples $\mathbf{x}$ and corresponding targets $\mathbf{y}$;
2. run the network on $\mathbf{x}$ (the **forward pass**) to obtain predictions $\hat{\mathbf{y}}$;
3. compute the loss of the network on the batch, a measure of the mismatch between $\hat{\mathbf{y}}$ and $\mathbf{y}$;
4. update all weights of the network in a way that slightly reduces the loss on this batch.

Repeat these steps in a loop, as often as necessary. Hopefully, the process will eventually converge on a network with very low training loss, which is to say that there will be a low mismatch between the predictions $\hat{\mathbf{y}}$ and the target $\mathbf{y}$. In the vernacular, we say that the ANN has **learned** to map its inputs to correct targets.

Step 1 is easy enough. Steps 2 and 3 are simply the application of a handful of tensor operations (or matrix multiplication, as above). Step 4 is more difficult: how do we update the network's weights? Given an individual weight coefficient in the network, how can we compute whether the coefficient should be increased or decreased, and by how much?

One solution is to successively minimize the objective function along coordinate directions to find the minimum of a function; this algorithm is called **coordinate descent** and at each iteration determines a coordinate, then minimizes over the corresponding hyperplane while fixing all other coordinates [6].

It is based on the idea that optimization can be achieved by minimizing along one direction at a time. Coordinate descent is useful in situations where the objective function is not **differentiable**, as is the case for most regularized regression models, say. But this approach would be inefficient in deep learning networks, where there is a large collection of individual weights to update. A smarter approach is use the fact that all operations used to propagate a signal in the network are differentiable, and compute the **gradient** of the objective function (loss) with regard to the network's coefficients.

Following a long-standing principle of calculus, we can decrease the objective function by updating the coefficients in the **opposite direction to the gradient**.[47] For an input vector $\mathbf{X}$, a weight matrix $\mathbf{W}$, a target $\mathbf{Y}$, and a loss function L, we predict a target candidate $\hat{\mathbf{Y}}(\mathbf{W})$, and compute the loss when approximating $\mathbf{Y}$ by $\hat{\mathbf{Y}}(\mathbf{W})$.

47: The gradient is the derivative of a tensor operation; it generalizes the notion of the derivative to functions of multidimensional inputs.

If $\mathbf{X}$ and $\mathbf{Y}$ are fixed, the loss function maps weights $\mathbf{W}$ to loss values: $f(\mathbf{W}) = L(\hat{\mathbf{Y}}(\mathbf{W}), \mathbf{Y})$.

In much the same way that the derivative of a univariate function $f(x)$ at a point x_0 is the slope of the tangent at f at x_0, the gradient $\nabla f(\mathbf{W}_0)$ is the tensor describing the **curvature** of $f(\mathbf{W})$ around $\mathbf{W}_0$. As is the case with the derivative, we can reduce $f(\mathbf{W})$ by moving $\mathbf{W}_0$ to

$$\mathbf{W}_1 = \mathbf{W}_0 - s\nabla f(\mathbf{W}_0),$$

where s is the **learning rate**, a small scalar needed to approximate the curvature of the hypersurface close to $\mathbf{W}_0$.

Stochastic Gradient Descent When dealing with ANNs, we can take advantage of the differentiability of the gradient by finding its **critical points** $\nabla f(\mathbf{W}) = 0$ analytically.

If the neural network contains Q edges, this requires solving a polynomial equation in Q variables. However, real-world ANNs often have over a few thousand such connections (if not more), and so this analytical approach is not reasonable.

Instead, we modify the parameters slightly based on the current loss value on a random batch of data. Since we are dealing with a differentiable function, we can use a **mini-batch stochastic gradient descent** (minibatch SGD) to update the weights, simply by modifying Step 4 of the gradient descent algorithm as follows:

4a. compute the gradient of the loss with regard to the weights (the **backward pass**);

4b. update the weights "a little" in the direction opposite the gradient.

Figure 21.23 illustrates how SGD works when the network only has the one parameter to learn, with a single training sample.

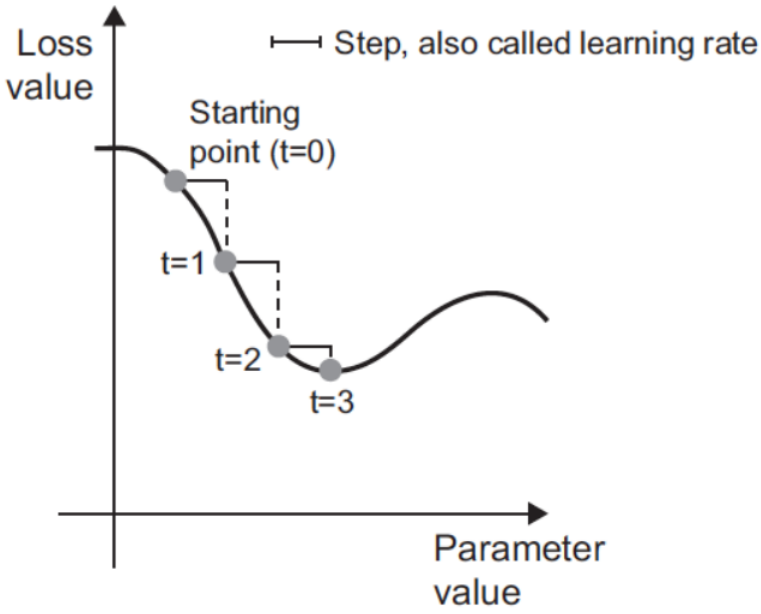


Figure 21.23: SGD with one parameter [6].

We automatically see why it is important to choose a reasonable learning rate (the step size); too small a value leads to either slow convergence or running the risk of staying stuck at some local minimum; too large a value may send the descent to essentially random locations on the curve and overshooting the global minimum altogether.

SGD Challenges The main issue with minibatch SGD is that "good" convergence rates are not guaranteed, but there are other challenges:

- selecting a reasonable learning rate can be difficult. Too small a rate leads to painfully slow convergence, too large a rate can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge [6];
- the same learning rate applies to all parameter updates, which might not be ideal when the data is sparse;
- a key challenge is in minimizing highly non-convex loss functions that commonly occur in ANNs and avoiding getting trapped in sub-optimal local minima or saddle points. It is hard for SGD to escape these sub-optimal local minima and even wors for the saddle points [8].

SGD Variants There are several SGD variants that are commonly used by the deep learning community to overcome the aforementioned challenges. They take into account the previous weight updates when computing the next weight update, rather than simply considering the current value of the gradients. Popular **optimzers** include SGD with momentum, Nesterov accelerated gradient, Adagrad, Adadelta, RMSProp, and many more [7, 24].[48]

48: A beautiful animation (created by A. Radford) compares the performance of different optimization algorithms and shows that the methods usually take different paths to reach the minimum.

ANNs can be quite accurate when making predictions – more than other algorithms, if given a proper set up (but this can be hard to achieve). They degrade gracefully, and they often work when other things fail:

- when the relationship between attributes is **complex**;
- when there are a lot of dependencies/**nonlinear relationships**;
- when the inputs are **messy** and highly-connected (images, text and speech), and
- when dealing with non-linear classification.

But they are relatively slow and prone to overfitting (unless they have access to large and diverse training sets), they are notoriously hard to interpret due to their **blackbox** nature, and there is no algorithm in place to select the optimal network topology.

Finally, even when they do perform better than other options, ANNs may not perform that much better due to the **No Free-Lunch** theorems; and they always remain susceptible to various forms of **adversarial attacks** [13], so they should be used with caution.[49]

49: For now, at least... who knows what the future holds.

Example: Wine Dataset In this example, we explore the `wine` dataset with the ANN architecture implemented in the `R` package `neuralnet`. We will revisit deep learning networks, and more complicated topologies, in Chapter 31.

```
wine = read.csv("wine.csv", header=TRUE)
wine = as.data.frame(wine)
str(wine)
```

```
'data.frame':   178 obs. of  14 variables:
 $ Class               : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Alcohol             : num  14.2 13.2 13.2 14.4 13.2 ...
 $ Malic.acid          : num  1.71 1.78 2.36 1.95 2.59 ...
 $ Ash                 : num  2.43 2.14 2.67 2.5 2.87 ...
 $ Alcalinity.of.ash   : num  15.6 11.2 18.6 16.8 21 ...
 $ Magnesium           : int  127 100 101 113 118 112 ...
 $ Total.phenols       : num  2.8 2.65 2.8 3.85 2.8 ...
 $ Flavanoids          : num  3.06 2.76 3.24 3.49 2.69 ...
 $ Nonflavanoid.phenols: num  0.28 0.26 0.3 0.24 0.39 ...
 $ Proanthocyanins     : num  2.29 1.28 2.81 2.18 1.82 ...
 $ Colour.intensity    : num  5.64 4.38 5.68 7.8 4.32 ...
 $ Hue                 : num  1.04 1.05 1.03 0.86 1.04 ...
 $ OD280.OD315         : num  3.92 3.4 3.17 3.45 2.93 ...
 $ Proline             : int  1065 1050 1185 1480 735 ...
```

```
table(wine$Class)
```

```
 1  2  3
59 71 48
```

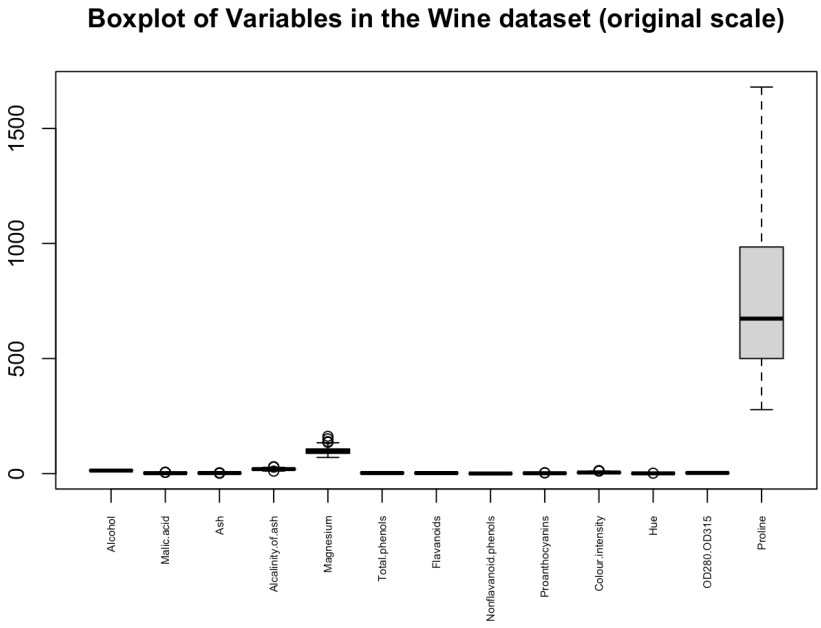
We notice that there are only 3 classes: 1, 2, 3. These classes will be the level of the categorical response variable for a classification task.

We set-up the model parameters/inputs as follows:

```
# Number of instances
n = nrow(wine)
# Dependent variable - Class
Y = wine$Class
# Independent variables (full)
X = wine[,-1]

# Indices for Class=1,2,3
C1.loc = which(Y==1)
C2.loc = which(Y==2)
C3.loc = which(Y==3)
```

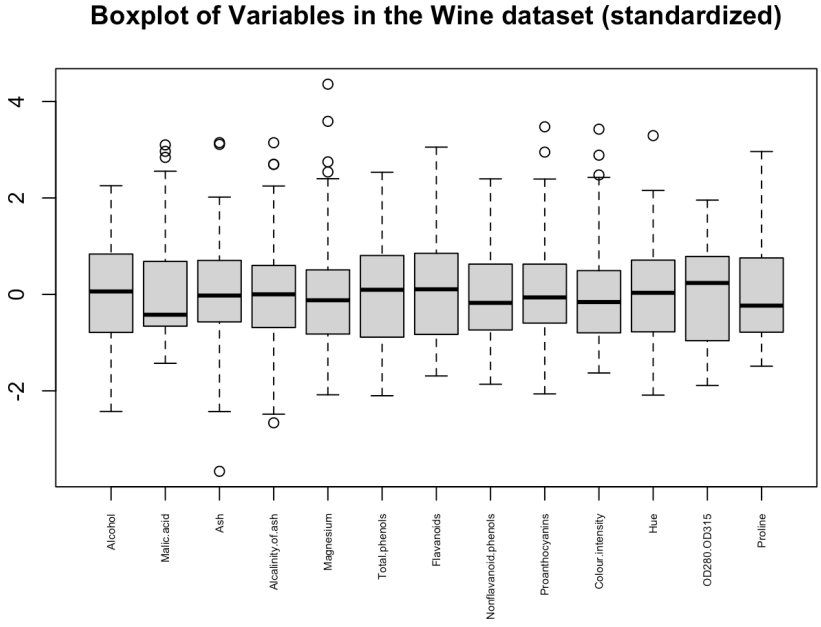
We begin data exploration by taking a look at the variables' boxplots, an excellent way to understand the distribution of each variable.

```
plot.title = "Boxplot of Variables in the Wine dataset
              (original scale)"
boxplot(X, main=plot.title, xaxt="n")
axis(1,at=1:dim(X)[2],labels=colnames(X),las=2,cex.axis=0.5)
```



We clearly see that `Proline` has higher magnitudes in mean value and variability. This suggests that if we apply reduction techniques like PCA to the wine dataset, the 1st principal component will be based almost entirely on the `Proline` value. In order to reduce undue effects, we need to standardize the data first (see Chapter 23 for details).

```
# Standardized predictor variables (full)
X.std = scale(X)

plot.title = "Boxplot of Variables in the Wine dataset
              (standardized)"
boxplot(X.std, main=plot.title, xaxt="n")
axis(1,at=1:dim(X)[2],labels=colnames(X),las=2,cex.axis=0.5)
```

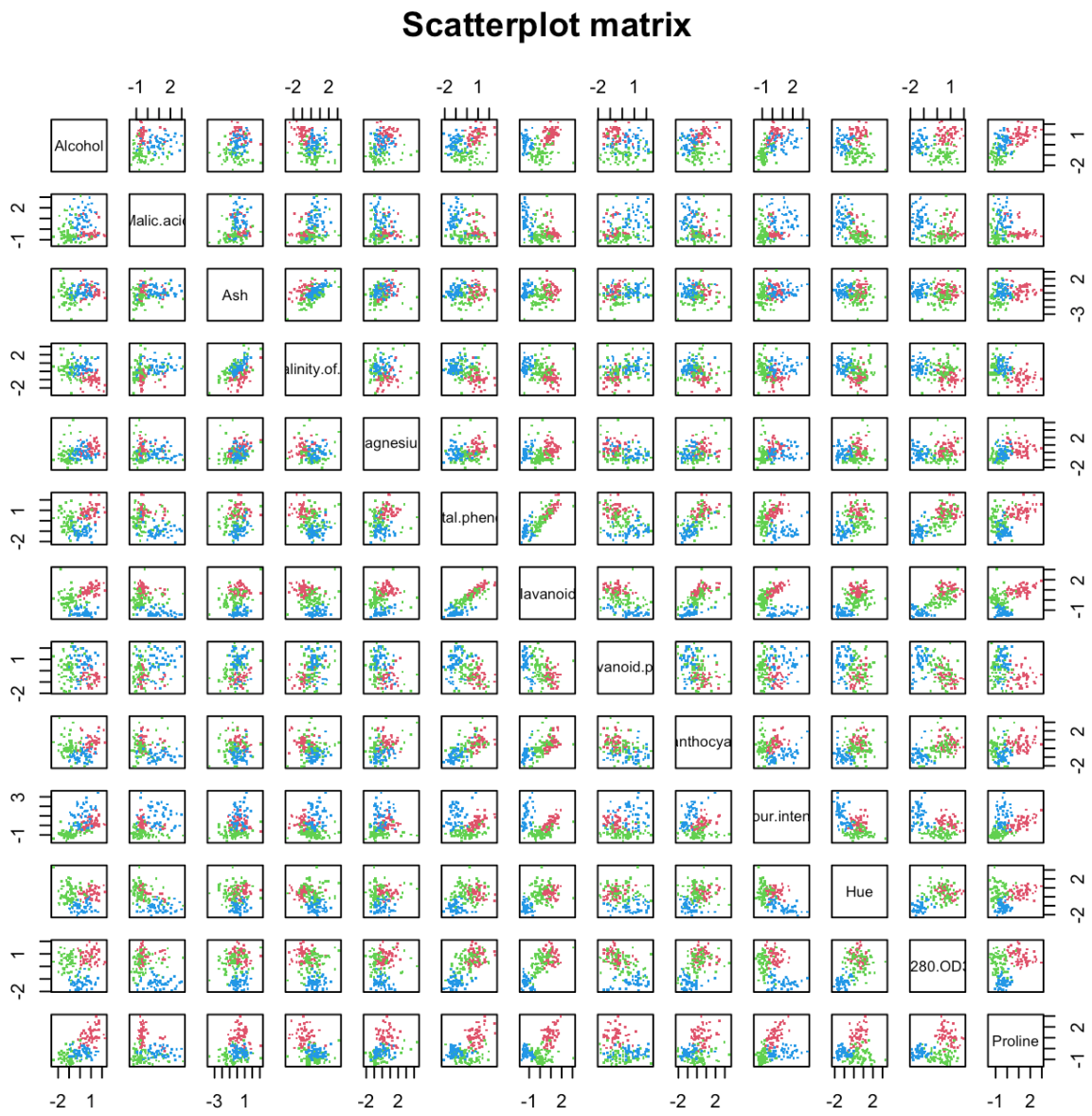


We now shift our focus on understanding the relationships amongst the explanatory variables. This is important because:

- scatter plots show us whether classification is a relatively simple task for our data;
- they let us visually inspect potential outliers or influential points;
- correlations amongst variables tell us whether it is necessary to retain all of them, and
- the variance inflation factor (VIF) helps us determine which variables can be removed in order to obtain more stable models, etc.

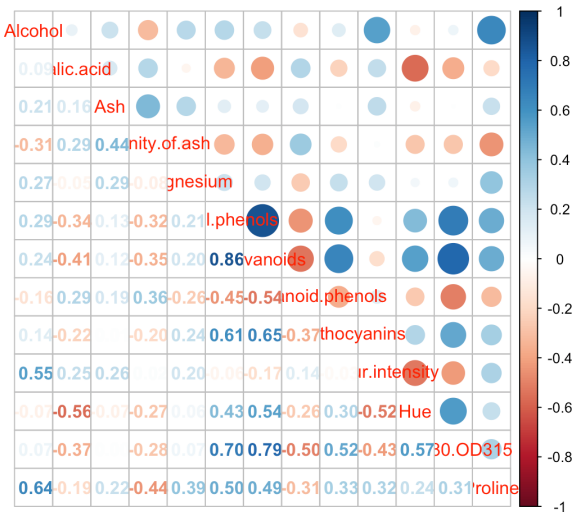
Let us first take a look at the scatterplot matrix:

```
plot.title = "Scatterplot matrix"
pairs(X.std, main=plot.title, col=Y+1, cex=0.5, pch=".")
```



We can also calculate and display the correlation matrix:

```
X.cor = cor(X.std)
corrplot::corrplot.mixed(X.cor)
```



We write a little function that will compute the VIF of each variable in a design matrix

```
vif <- function(X){
  vif=rep(0,dim(X)[2])
  for (i in 1:dim(X)[2]){
  expl=X[,-c(i)]
  y=lm(X[,i]~expl)
  vif[i]=1/(1-summary(y)$r.squared)}
  return(vif)
}

vif.X = matrix(vif(X.std), nrow=1)
colnames(vif.X) = colnames(X)
rownames(vif.X) = "VIF"
round(t(vif.X),2)
```

	VIF		VIF
Alcohol	2.46	Nonflavanoid.phenols	1.80
Malic.acid	1.66	Proanthocyanins	1.98
Ash	2.19	Colour.intensity	3.03
Alcalinity.of.ash	2.24	Hue	2.55
Magnesium	1.42	OD280.OD315	3.79
Total.phenols	4.33	Proline	2.82
Flavanoids	7.03		

We see that `Flavonoids` has high multicollinearity with respect to the other variables, as its VIF is greater than 5; as such we have reasonable grounds to remove that variable from further analyses as the other variables can explain how `Flavonoids` would behave and doing so might reduce the risk of creating **unstable models**.

```
X = X[,-7]
X.std = X.std[,-7]
```

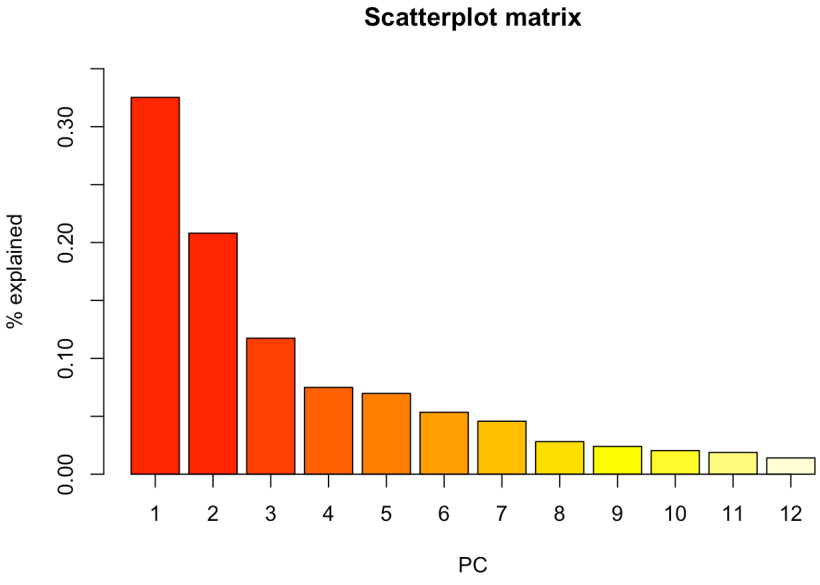
Now we apply a PCA reduction (see Chapter 23 for details) to further reduce the problem complexity. We start by performing principal component analysis on the standardized data:

```
pca.std = prcomp(X.std)
summary(pca.std)
```

```
Importance of components:
                          PC1    PC2    PC3     PC4
Standard deviation     1.9757 1.5802 1.1870 0.94820
Proportion of Variance 0.3253 0.2081 0.1174 0.07492
Cumulative Proportion  0.3253 0.5334 0.6508 0.72569

                          PC5     PC6     PC7     PC8
Standard deviation     0.91472 0.80087 0.74082 0.58095
Proportion of Variance 0.06973 0.05345 0.04573 0.02813
Cumulative Proportion  0.79541 0.84886 0.89460 0.92272

                          PC9    PC10   PC11    PC12
Standard deviation     0.53687 0.49487 0.4750 0.41059
Proportion of Variance 0.02402 0.02041 0.0188 0.01405
Cumulative Proportion  0.94674 0.96715 0.9859 1.00000
```

The scree plot for proportions of variance explained by each PC is:

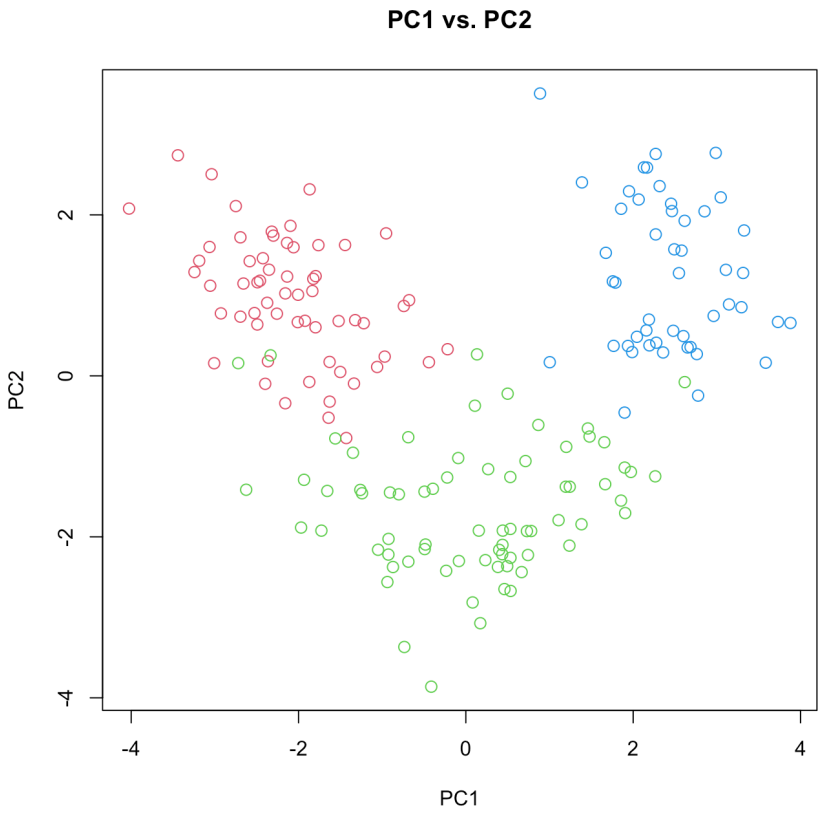
```
scree.y = eigen(t(X.std)%*%X.std)$values/
                sum(eigen(t(X.std)%*%X.std)$values)
barplot(scree.y, main=plot.title,ylim=c(0, 0.35),
        ylab="% explained", xlab="PC",col=heat.colors(12))
test = seq(0.7, 13.9, length.out=12)
axis(1, at=test, labels=1:12)
```



Based on the summary table, the first 6 PC would be required to retain 80% of the explanatory power; the scree plot, on the other hand, shows a knee at the 4th component.

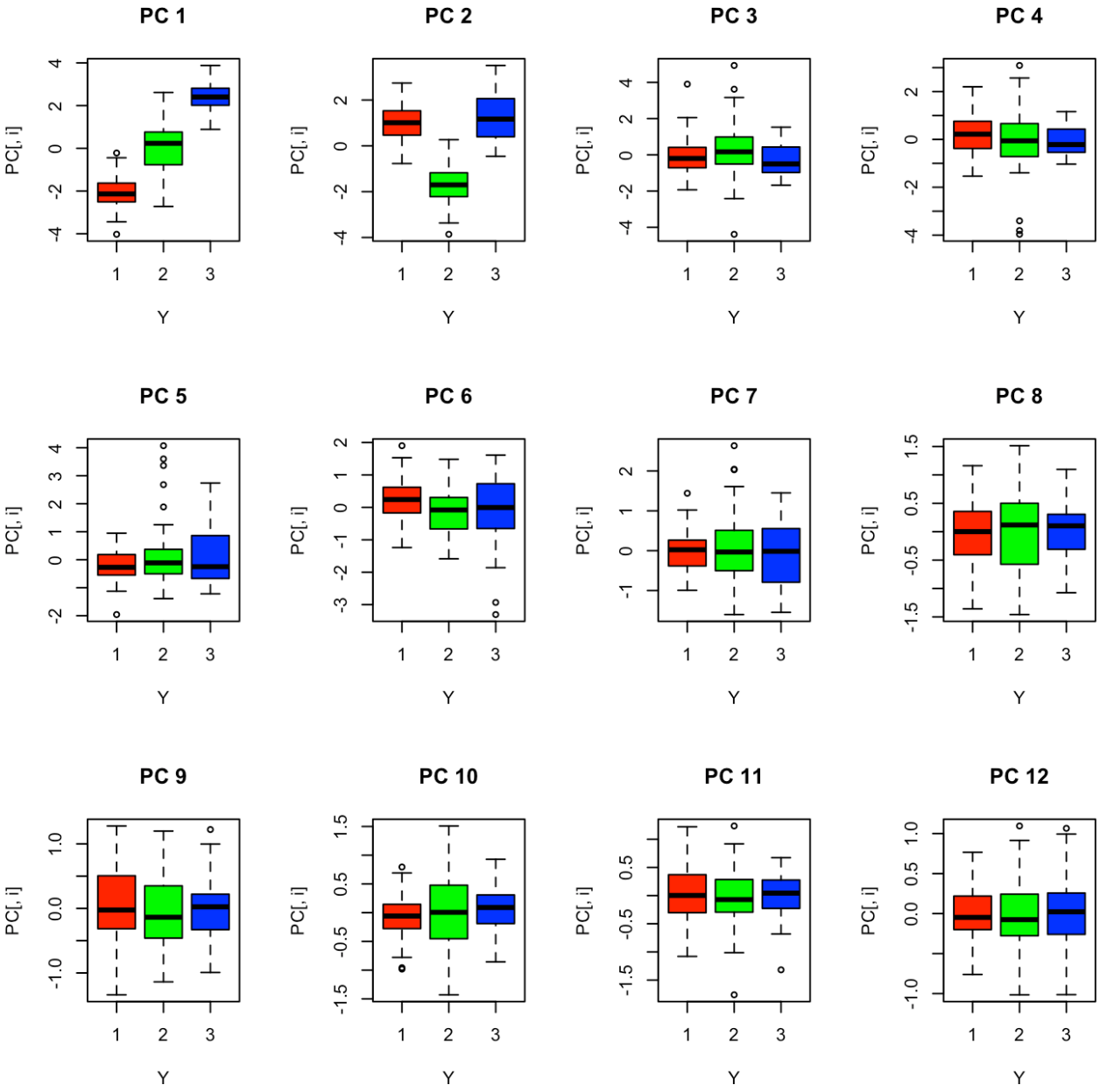
We can explore this further: let us take a look at the scatterplot of the first two PC. We start by transforming `X.std` into its principal coordinates:

```
PC = X.std %*% pca.std$rotation
PCnames = c("PC1","PC2","PC3","PC4","PC5","PC6","PC7",
            "PC8","PC9","PC10","PC11","PC12")
colnames(PC) <- PCnames

plot.title = "PC1 vs. PC2"
plot(PC[,1], PC[,2], cex=1.2, main=plot.title, col=Y+1,
     xlab="PC1", ylab="PC2")
```



The scatterplot shows that the three classes are separated reasonably well by PC1 and PC2 (although, not linearly).

```
plot.title = "Boxplots (in Principal Coordinates)"
  par(mfrow = c(3,4))
  for (i in 1:12){
  plot.title.ind = paste("PC ", i, sep="")
    boxplot(PC[,i]~Y, main=plot.title.ind,
            col=c("red","green","blue"))
  }
```


PC 1
PC 2
PC 3
PC 4
PC 5
PC 6
PC 7
PC 8
PC 9
PC 10
PC 11
PC 12
PC[, i]
Y

The boxplots further leave the impression that PC3 to PC12 do not provide as clear a separation as do PC1 and PC2. We will thus use only the latter to **visually** evaluate the effectiveness of ANN (and its prediction regions).

In order to evaluate the effectiveness of the model (i.e., does it have good predictive power without overfitting the data?), we split the data into training and testing sets.

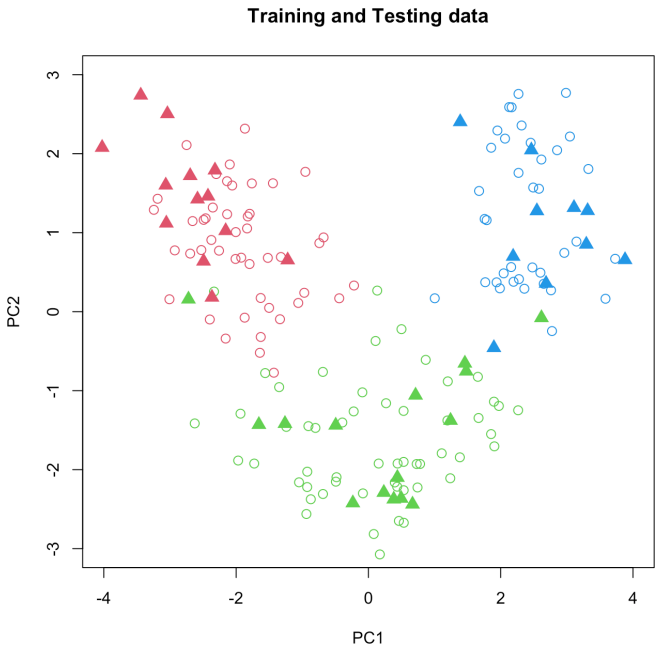
The model is then developed using the training set (i.e., optimized using a subset of data), and then evaluated for its prediction power using the testing set.

There are $n = 178$ observations in total: we sample 140 of them for the training set (say, 46 of class 1, 56 of class 2, and 38 of class 3). The remaining 38 observations form the testing set.

```
set.seed(1111) # for replicabilty
C1.train.loc = sort(sample(C1.loc, size=46))
C2.train.loc = sort(sample(C2.loc, size=56))
C3.train.loc = sort(sample(C3.loc, size=38))
train.loc = c(C1.train.loc, C2.train.loc, C3.train.loc)
test.loc = which(!(1:length(Y) %in% train.loc))

# training data
PC.train = PC[train.loc,]
Y.train = Y[train.loc]
dat.train = as.data.frame(cbind(nnet::class.ind(Y.train),
                                PC.train))
colnames(dat.train)[1:3] = c("C1", "C2", "C3")

# testing data
PC.test = PC[test.loc,]
Y.test = Y[test.loc]
dat.test = as.data.frame(cbind(nnet::class.ind(Y.test),
                               PC.test))
colnames(dat.test)[1:3] = c("C1", "C2", "C3")
```

We display the training dataset (circles) and testing dataset (triangles) on the PC1/PC2 scatterplot.

```
plot.title = "Training and Testing data"
xlimit = c(-4,4)
ylimit = c(-3,3)

plot(dat.train$PC1, dat.train$PC2, cex=1.2, col=Y.train+1,
     main=plot.title, xlab="PC1", ylab="PC2", xlim=xlimit,
     ylim=ylimit)
points(dat.test$PC1, dat.test$PC2, pch=17, cex=1.5,
       col=Y.test+1)
```



If we only use the PC1/PC2-reduced data, we would expect that at least two of the test observations would be misclassified (the left-most and right-most green triangles, respectively).

We are finally ready to build an ANN *via* the `R` package `neuralnet` (the main function is also called `neuralnet()`). We run three analyses:

1. using only the first two principal components as inputs;
2. useing the first six principal components as inputs, and
3. using all 12 principal components as inputs.

We start by forming a grid in the PC1/PC2 space on which we can colour the prediction regions.

```
predict.region.PC1=seq(-5,5, length.out=100)
predict.region.PC2=seq(-4,4, length.out=100)
predict.region=expand.grid(x=predict.region.PC1,
                           y=predict.region.PC2)
```

We will also use an expanded form of the confusion matrix:

```
# A souped-up version of the confusion matrix
confusion.expand <- function (pred.c, class) {
   temp <-mda::confusion(pred.c,class)
   row.sum <- apply(temp,1,sum)
   col.sum <- apply(temp,2,sum)
   t.sum <- sum(col.sum)
   tmp <- rbind(temp, rep("----", dim(temp)[2]), col.sum)
   tmp <- noquote(cbind(tmp, rep("|",dim(tmp)[1]),
                  c(row.sum, "----", t.sum)))
   dimnames(tmp)<-list(object =
            c(dimnames(temp)[[1]],"-------","Col Sum"),
            true = c(dimnames(temp)[[2]],"|","Row Sum"))
```

```
   attr(tmp, "error") <- attr(temp, "error")
   attr(tmp, "mismatch") <- attr(temp, "mismatch")
   return(tmp)
}
```

In what follows, we build an ANN model using `neuralnet`, with PC1 and PC2 as inputs. The parameter `model.structure`, which defines the number of hidden nodes in each hidden layer, is modifiable:

- a model with no hidden layer would have `model.structure = 0`;
- 1 hidden layer of 3 nodes would require `model.structure = 3`;
- 2 hidden layers of 5 and 10 nodes, respectively, would require `model.structure = c(5,10)`, and so on.

We will use 2 hidden layers of 10 nodes each.

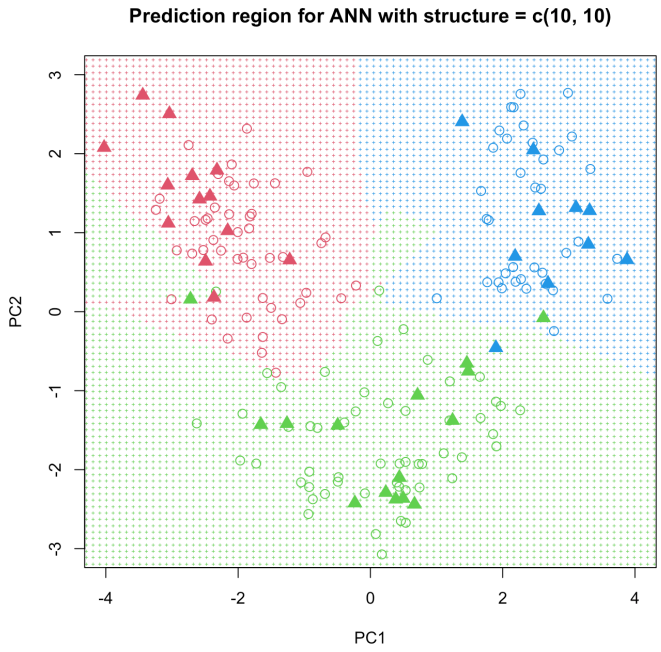
```
model.structure = c(10,10)
model1 <- neuralnet::neuralnet(C1 + C2 + C3 ~ PC1 + PC2,
            data = dat.train, hidden = model.structure,
            err.fct = "ce", linear.output = FALSE)
prob.model1.test <- neuralnet::compute(model1, PC.test[,1:2])
predict.model1.test = max.col(prob.model1.test$net.result)
print(paste("Confusion matrix (testing) for model = ",
            list(model.structure)[1], sep=""))
(conf.test=confusion.expand(predict.model1.test, Y.test))
```

```
[1] "Confusion matrix (testing) for model = c(10, 10)"
         true
object    1    2    3    | Row Sum
  1       13   1    0    | 14
  2       0    13   1    | 14
  3       0    1    9    | 10
  ------- ---- ---- ---- | ----
  Col Sum 13   15   10   | 38

attr(,"error")
[1] 0.07894737
```

We can compute the prediction region for the two-input model and display it as follows:

```
prob.model1.region <- neuralnet::compute(model1,
                                predict.region[,1:2])
predict.model1.region = max.col(prob.model1.region$net.result)
plot.title=paste("Prediction region for ANN with structure = ",
                 list(model.structure)[1], sep="")
plot(predict.region[,1], predict.region[,2],
     main=plot.title, xlim=xlimit, ylim=ylimit,
     xlab="PC1", ylab="PC2",
     col=predict.model1.region+1, pch="+", cex=0.4)
points(dat.train$PC1, dat.train$PC2, cex=1.2,
       col=Y.train+1)
```

```
points(dat.test$PC1, dat.test$PC2, pch=17, cex=1.5,
       col=Y.test+1)
```



Note the complex decision boundary.

Since the error function we seek to minimize (e.g., SSE) is **non-convex**, it is possible for ANN to get stuck at local minima, rather than converge to the global minimum. We can run the model a number of times (say, 50 replicates) and find the average prediction.[50]

50: **Note:** this code may produce an error saying that ANN has issues converging (especially for simpler models). If this happens, the simple solution is to re-run the code again or change the seed.

```
model.structure = c(10,10)
n.j = 50
conf.train.vector = conf.test.vector = NULL

for (j in 1:n.j){
    model1 <- neuralnet::neuralnet(C1 + C2 + C3 ~ PC1 + PC2,
    data = dat.train, hidden = model.structure,
    err.fct = "ce", linear.output = FALSE)

    prob.model1.test <- neuralnet::compute(model1,
                                           PC.test[,1:2])
    predict.model1.test = max.col(prob.model1.test$net.result)
    conf.test = confusion.expand(predict.model1.test,
                                 Y.test)
    conf.test.vector=c(conf.test.vector,
                       attributes(conf.test)$error)
}

# number of misclassifications
conf.test.vector = round(conf.test.vector*length(Y.test))

print(paste("Summary of number of misclassifications
      in testing data out of", n.j, "trials", sep=" "))
```

```
round(summary(conf.test.vector), digits=2)
```

```
[1] "Summary of number of misclassifications in testing data out of 50 trials"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2.00    3.00    3.00    2.92    3.00    4.00
```

We build ANN for the PCA-reduced 6-input dataset.[51] For each ANN topology, we replicate the process 25 times. It should be noted that prediction regions are not computed, as our input is in more than 2 dimensional space.

51: We will try 11 different topologies: no hidden layer; 1 hidden layer with 2, 6, 10, and 30 nodes; 2-hidden layers with (6,6), (10,10), (30,30) nodes, and 30-hidden layers with (6,6,6), (10,10,10) and (30,30,30) nodes.

```
model.structure = list(0,           # no hidden layer
    2, 6, 10, 30,                   # 1 hidden layer
    rep(6,2), rep(10,2), rep(30,2), # 2 hidden layers
    rep(6,3), rep(10,3), rep(30,3)) # 3 hidden layers

set.seed(1)
results = NULL
n.loop = length(model.structure)
n.j = 25

for (i in 1:n.loop){
  conf.train.vector = conf.test.vector = NULL

  for (j in 1:n.j){
    model1 <- neuralnet::neuralnet(C1 + C2 + C3 ~
            PC1 + PC2 + PC3 + PC4 + PC5 + PC6,
            data = dat.train, hidden = model.structure[[i]],
            err.fct = "ce", linear.output = FALSE)

    prob.model1.test <- neuralnet::compute(model1,
            PC.test[,1:6])
    predict.model1.test = max.col(prob.model1.test$net.result)
    conf.test=confusion.expand(predict.model1.test,
            Y.test)
    conf.test.vector=c(conf.test.vector,
            attributes(conf.test)$error)
  }

  results[[i]] = summary(round(conf.test.vector*length(Y.test)))
}

results = as.data.frame(dplyr::bind_rows(results,
                        .id = "column_label"))
colnames(results) <- c("hidden", "min", "Q1", "med", "mean",
                        "Q3", "max")
results$hidden <- model.structure
```

hidden	min	Q1	med	mean	Q3	max
0	2	2	2	2.00	2	2
2	1	2	2	1.92	2	3
6	1	2	2	1.92	2	2

```
10          1   2   2   1.96   2   2
30          2   2   2   2.00   2   2
6, 6        1   2   2   1.96   2   2
10, 10      1   2   2   1.88   2   2
30, 30      2   2   2   2.00   2   2
6, 6, 6     1   2   2   1.88   2   2
10, 10, 10  1   2   2   1.88   2   2
30, 30, 30  1   2   2   1.96   2   2
```

We can repeat this process once more, using all 12 PC.

```
for (i in 1:n.loop){
  conf.train.vector = conf.test.vector = NULL

  for (j in 1:n.j){
    model1 <- neuralnet::neuralnet(C1 + C2 + C3 ~ .,
                data = dat.train,
                hidden = model.structure[[i]],
                err.fct = "ce", linear.output = FALSE)

    prob.model1.test <- neuralnet::compute(model1,
                                           PC.test[,1:12])
    predict.model1.test = max.col(prob.model1.test$net.result)
    conf.test=confusion.expand(predict.model1.test,
                               Y.test)
    conf.test.vector=c(conf.test.vector,
                       attributes(conf.test)$error)
  }

  results[[i]] = summary(round(conf.test.vector*length(Y.test)))
}

results = as.data.frame(dplyr::bind_rows(results,
                        .id = "column_label"))
colnames(results) <- c("hidden", "min", "Q1", "med", "mean",
                       "Q3", "max")
results$hidden <- model.structure
```

```
hidden      min Q1  med mean   Q3   max
0           2   2   2   2.00   2   2
2           1   1   1   1.80   3   4
6           1   2   2   2.04   2   3
10          1   2   2   1.96   2   2
30          2   2   2   2.00   2   2
6, 6        1   2   2   2.00   2   3
10, 10      1   2   2   1.92   2   3
30, 30      1   2   2   1.84   2   2
6, 6, 6     1   1   2   1.76   2   3
10, 10, 10  1   2   2   1.84   2   2
30, 30, 30  1   1   2   1.68   2   2
```

Comparing the mean number of misclassifications, what can we conclude?

21.4.4 Naïve Bayes Classifiers

In classical statistics, model parameters (such as μ and σ, say) are treated as constants; **Bayesian statistics**, on the other hand assume that model parameters are **random variables**.

Bayes' Theorem lies at the foundation of this approach:

$$P(\text{hypothesis} \mid \text{data}) = \frac{P(\text{data} \mid \text{hypothesis}) \times P(\text{hypothesis})}{P(\text{data})},$$

or simply

$$P(H \mid D) = \frac{P(D \mid H) \times P(H)}{P(D)}.$$

This is sometimes written in shorthand as $P(H \mid D) \propto P(D \mid H) \times P(H)$; in other words, our **degree of belief in a hypothesis should be updated by the evidence provided by the data**.[52]

52: Nobody disputes the validity of Bayes' Theorem, and it has proven to be a useful component in various models and algorithms, such as email spam filters, and the following example, but the **use** of Bayesian statistics is controversial in many quarters. We will discuss Bayesian data analysis in depth in Chapter 25.

Naïve Bayes Classification for Tumor Diagnoses Suppose we are interested in diagnosing whether a tumor is benign or malignant, based on several measurements obtained from video imaging.

Bayes' Theorem can be recast as

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}} \propto \text{likelihood} \times \text{prior},$$

where

- **posterior:** $P(H \mid D)$ = based on collected data, how likely is a given tumor to be benign (or malignant)?
- **prior:** $P(H)$ = in what proportion are tumors benign (or malignant) in general?
- **likelihood:** $P(D \mid H)$ = knowing a tumor is benign (or malignant), how likely is it that these particular measurements would have been observed?
- **evidence:** $P(D)$ = regardless of a tumor being benign or malignant, what is the chance that a tumor has the observed characteristics?

The **naïve Bayes classifiers** (NBC) procedure is straightforward.

1. **Objective function:** a simple way to determine whether a tumor is benign or malignant is to compare **posterior probabilities** and choose the one with highest probability. That is, we diagnose a tumor as **malignant** if

$$\frac{P(\text{malignant} \mid D)}{P(\text{benign} \mid D)} = \frac{P(D \mid \text{malignant}) \times P(\text{malignant})}{P(D \mid \text{benign}) \times P(\text{benign})} > 1,$$

   and as **benign** otherwise.
2. **Dataset:** there are $n = 699$ observations (from the [in]famous BreastCancer-Wisconsin.csv dataset) with nine predictor measurements, each scored on a scale of 1 to 10, a score of 0 being reserved for missing values. The predictors include items such as `Clump_Thickness` and `Bare_Nuclei`; the categorical response variable is the `Class` (`Benign`, `Malignant`).

```
dat.BC = read.csv("BreastCancer-Wisconsin.csv",
            header=TRUE, stringsAsFactors = TRUE)
str(dat.BC)
```

```
'data.frame':   699 obs. of  10 variables:
 $ Clump_Thickness            : int  5 5 3 6 4 8 1 2 2 4 ...
 $ Uniformity_of_Cell_Size    : int  1 4 1 8 1 10 1 1 1 2 ...
 $ Uniformity_of_Cell_Shape   : int  1 4 1 8 1 10 1 2 1 1 ...
 $ Marginal_Adhesion          : int  1 5 1 1 3 8 1 1 1 1 ...
 $ Single_Epithelial_Cell_Size: int  2 7 2 3 2 7 2 2 2 2 ...
 $ Bare_Nuclei                : int  1 10 2 4 1 10 10 1 1 1 ...
 $ Bland_Chromatin            : int  3 3 3 3 3 9 3 3 1 2 ...
 $ Normal_Nucleoli            : int  1 2 1 7 1 7 1 1 1 1 ...
 $ Mitoses                    : int  1 1 1 1 1 1 1 1 5 1 ...
 $ Class                      : Factor w/ 2 levels "Benign","Malignant": 1 1 1 1 1 2 1 1 1 1 ...
```

In table layout, the first 6 observations look like:

```
head(dat.BC)
```

Clump_Thickness	Uniformity_of_Cell_Size	Uniformity_of_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Size	Bare_Nuclei	Bland_Chromatin	Normal_Nucleoli	Mitoses	Class
5	1	1	1	2	1	3	1	1	Benign
5	4	4	5	7	10	3	2	1	Benign
3	1	1	1	2	2	3	1	1	Benign
6	8	8	1	3	4	3	7	1	Benign
4	1	1	3	2	1	3	1	1	Benign
8	10	10	8	7	10	9	7	1	Malignant

We create a training/testing split for the data, by selecting roughly 80%/20% of the observations.

```
set.seed(1234) # for reproducibility
ind = 1:nrow(dat.BC)
prop.train = 0.8
n.train = floor(nrow(dat.BC)*prop.train)

# indices of training observations
loc.train = sort(sample(ind, n.train, replace=FALSE))
# indices of testing observations
loc.test = ind[-which(ind %in% loc.train)]
# training data
dat.BC.train = dat.BC[loc.train,]
# test data
dat.BC.test = dat.BC[loc.test,]
```

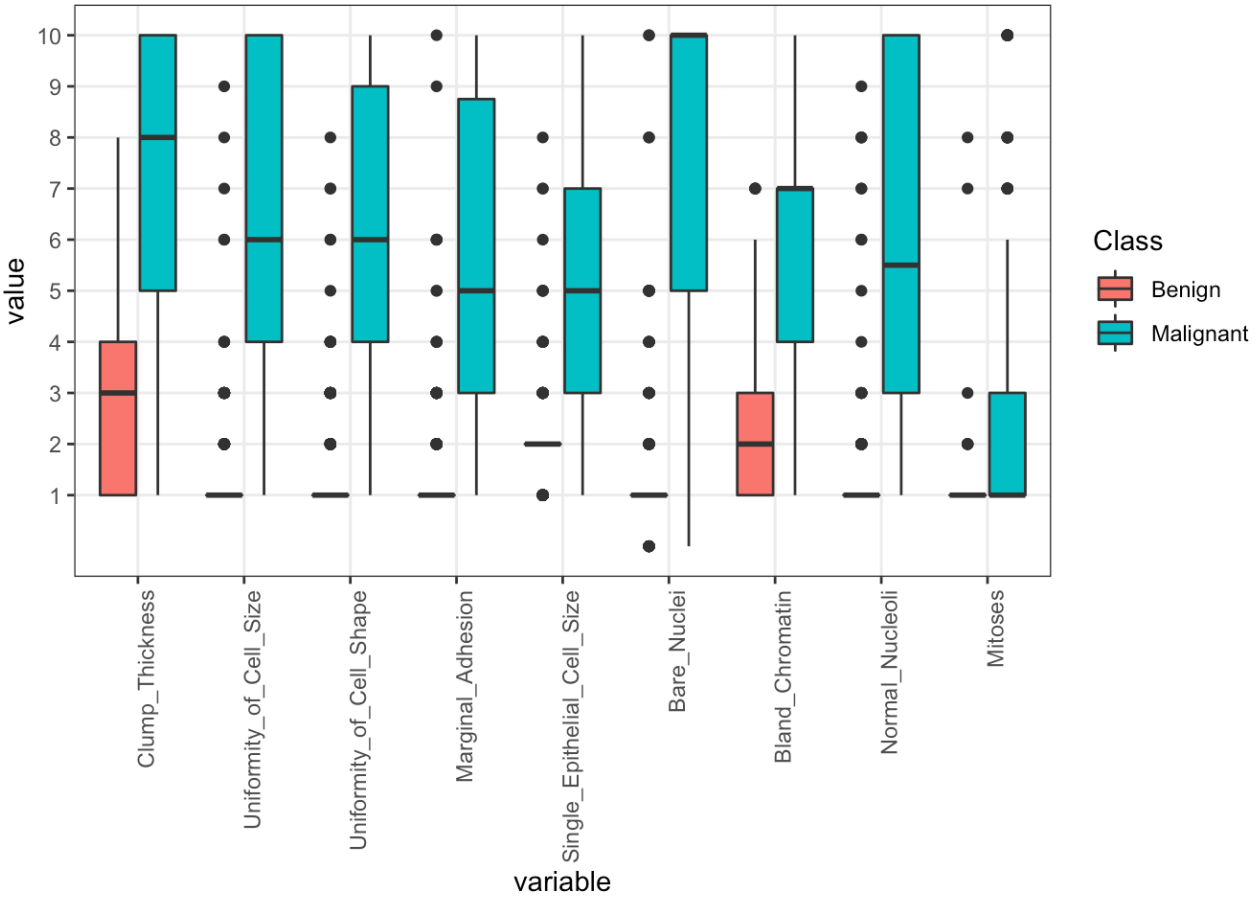
We separate the Benign and Malignant subsets of the training data for graphing purposes.

```
location.Benign = which(dat.BC.train$Class=="Benign")
location.Malignant = which(!(1:nrow(dat.BC.train)
        %in% location.Benign))
cols_remove = c("Class")
```

```
dat.Benign=dat.BC.train[location.Benign,!colnames(dat.BC)
        %in% cols_remove]
dat.Malignant=dat.BC.train[location.Malignant,
        !colnames(dat.BC) %in% cols_remove]
```

The boxplots of the training measurements are shown below.

```
library(gpplot2)
dat.BC2 = reshape2::melt(dat.BC.train, id.var="Class")
ggplot(data = dat.BC2, aes(x=variable, y=value)) +
  geom_boxplot(aes(fill=Class)) +
  scale_y_discrete(limits = 1:10) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



From these plots, we learn that benign tumors have lower scores on average, while malignant tumors have much higher scores and variabilities.[53]

3. **Assumptions:** we assume that the scores of each measurement in a class are independent of one another (hence the **naive** qualifier); this assumption reduces the likelihood to

$$\begin{aligned} P(H \mid D) &= P(H \mid x_1, x_2, \cdots, x_9) \propto P(x_1, x_2, \cdots, x_9 \mid H) \times P(H) \\ &= P(x_1 \mid H) \times \cdots \times P(x_9 \mid H) \times P(H). \end{aligned}$$

Note that this assumption of **conditional independence** is not usually satisfied.

53: In what follows, we treat the test observations as **undiagnosed cases**.

4. **Prior distribution:** we can ask subject matter experts to provide a rough estimate for the general ratio of benign to malignant tumors, or use the proportion of benign tumors in the sample as our prior.[54] In this example, we will assume that the training data represents the tumor population adequately, and we use the observed proportions as estimated prior probabilities.

54: In situations where we have no knowledge about the distribution of priors, we may simply assume a **non-informative prior**. In this case, the prevalence rates would be the same for both responses.

```
n.Benign.train = nrow(dat.Benign)
n.Malignant.train = nrow(dat.Malignant)
(prior.Benign = n.Benign.train/(n.Benign.train +
                  n.Malignant.train))
(prior.Malignant = 1 - prior.Benign)
```

```
[1] 0.6529517
[1] 0.3470483
```

5. **Computation of likelihoods:** under conditional independence, each measurement is assumed to follow a multinomial distribution (since scores are provided on a 1 to 10 scale): for each predictor, for each class, we must estimate $p_1, \ldots, p_{10}$, with $p_1 + \cdots + p_{10} = 1$.[55] The best estimates are thus

55: In other problems, the predictors could be continuous rather than discrete, in which case we would use continuous distributions instead; even in discrete case, the multinomial assumption might not be appropriate.

$$\hat{p}_{\ell,\text{pred}} = \frac{\#\text{ of training cells in the class with pred score } \ell}{\#\text{ of training cells in the class}}, \quad \ell = 1, \ldots, 10.$$

This is done in the code below; note that `count.xyz` is a count matrix, while `freq.xyz` is a frequency matrix.

```
# Benign cells
count.Benign = freq.Benign = NULL
for (i in 1:(ncol(dat.BC.train)-1)){
  test.count = table(c(dat.Benign[,i],0:10))-1
  test.freq = test.count/sum(test.count)
  count.Benign = cbind(count.Benign, test.count)
  freq.Benign = cbind(freq.Benign, test.freq)
}
colnames(count.Benign) = colnames(freq.Benign)
         = colnames(dat.Benign)
p.Benign = freq.Benign
p.Benign[1,] = 1

# Malignant cells
count.Malignant = freq.Malignant = NULL
for (i in 1:(ncol(dat.BC.train)-1)){
  test.count = table(c(dat.Malignant[,i],0:10))-1
  test.freq = test.count/sum(test.count)
  count.Malignant = cbind(count.Malignant, test.count)
  freq.Malignant = cbind(freq.Malignant, test.freq)
}
colnames(count.Malignant) = colnames(freq.Malignant)
         = colnames(dat.Malignant)
p.Malignant = freq.Malignant
p.Malignant[1,] = 1
```

These are then the best estimates for the multinomial parameters, for the benign tumors:

```
table(p.Benign)
```

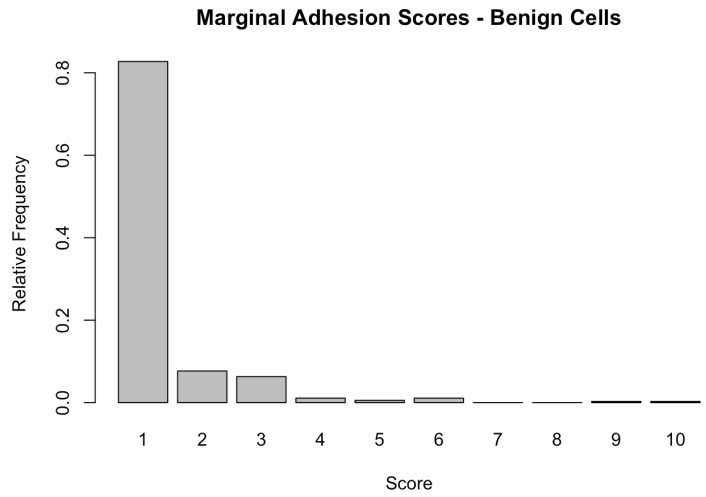
	Clump_Thickness	Uniformity_of_Cell_Size	Uniformity_of_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Size	Bare_Nuclei	Bland_Chromatin	Normal_Nucleoli	Mitoses
0	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000
1	0.3342466	0.8356164	0.7616438	0.8273973	0.1123288	0.8383562	0.3397260	0.8931507	0.9780822
2	0.0986301	0.0821918	0.1178082	0.0767123	0.7863014	0.0465753	0.3589041	0.0575342	0.0136986
3	0.2000000	0.0575342	0.0821918	0.0630137	0.0630137	0.0356164	0.2657534	0.0219178	0.0027397
4	0.1315068	0.0109589	0.0219178	0.0109589	0.0164384	0.0136986	0.0136986	0.0027397	0.0000000
5	0.1945205	0.0000000	0.0027397	0.0054795	0.0109589	0.0219178	0.0082192	0.0027397	0.0000000
6	0.0356164	0.0054795	0.0054795	0.0109589	0.0027397	0.0000000	0.0027397	0.0054795	0.0000000
7	0.0027397	0.0027397	0.0054795	0.0000000	0.0054795	0.0000000	0.0109589	0.0054795	0.0027397
8	0.0027397	0.0027397	0.0027397	0.0000000	0.0027397	0.0054795	0.0000000	0.0082192	0.0027397
9	0.0000000	0.0027397	0.0000000	0.0027397	0.0000000	0.0000000	0.0000000	0.0027397	0.0000000
10	0.0000000	0.0000000	0.0000000	0.0027397	0.0000000	0.0054795	0.0000000	0.0000000	0.0000000

For the malignant tumors, we have:

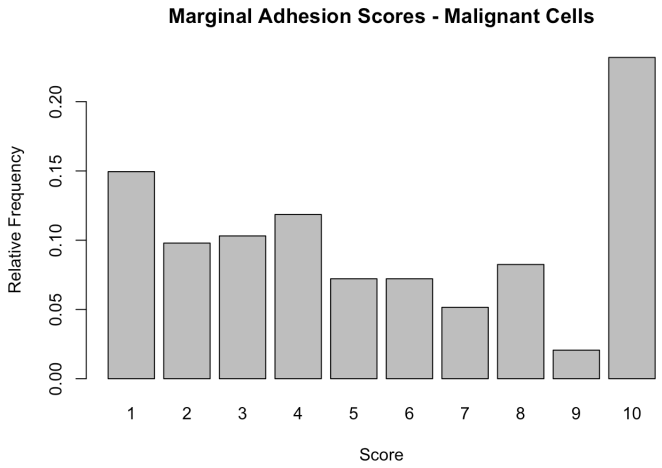
```
table(p.Malignant)
```

	Clump_Thickness	Uniformity_of_Cell_Size	Uniformity_of_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Size	Bare_Nuclei	Bland_Chromatin	Normal_Nucleoli	Mitoses
0	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000
1	0.0154639	0.0103093	0.0103093	0.1494845	0.0051546	0.0463918	0.0051546	0.1752577	0.5257732
2	0.0154639	0.0360825	0.0257732	0.0979381	0.0927835	0.0257732	0.0309278	0.0309278	0.1082474
3	0.0567010	0.1134021	0.0979381	0.1030928	0.2010309	0.0721649	0.1546392	0.1391753	0.1340206
4	0.0567010	0.1391753	0.1546392	0.1185567	0.1391753	0.0567010	0.1185567	0.0773196	0.0515464
5	0.1649485	0.1185567	0.1185567	0.0721649	0.1494845	0.0721649	0.1340206	0.0773196	0.0257732
6	0.0618557	0.1030928	0.1030928	0.0721649	0.1546392	0.0206186	0.0360825	0.0721649	0.0154639
7	0.0927835	0.0824742	0.1082474	0.0515464	0.0360825	0.0309278	0.2783505	0.0463918	0.0412371
8	0.1701031	0.1134021	0.1082474	0.0824742	0.0824742	0.0876289	0.1082474	0.0773196	0.0309278
9	0.0618557	0.0154639	0.0360825	0.0206186	0.0103093	0.0360825	0.0463918	0.0412371	0.0000000
10	0.3041237	0.2680412	0.2371134	0.2319588	0.1288660	0.5463918	0.0876289	0.2628866	0.0670103

```
barplot(p.Benign[2:11,4],
  xlab = "Score",  ylab = "Relative Frequency",
  main = "Marginal Adhesion Scores - Benign Cells")
```



```
barplot(p.Malignant[2:11,4],
  xlab = "Score", ylab = "Relative Frequency",
  main = "Marginal Adhesion Scores - Malignant Cells")
```



Multiplying probabilities across predictors from each multinomial distribution (one each for both classes) provides the overall likelihoods for benign and malignant tumors, respectively.
For instance, if the signature of an undiagnosed case was

$$(1, 1, 3, 1, 0, 2, 3, 2, 2),$$

then we would multiply the probabilities corresponding to each score across predictors, once assuming that the cell was benign, and once assuming it was malignant:

$$\begin{aligned} P(D \mid H) &= P(\mathbf{x} = (1,1,3,1,0,2,3,2,2) \mid H) \\ &= P(x_1 = 1 \mid H) \times P(x_2 = 1 \mid H) \times P(x_3 = 3 \mid H) \\ &\quad \times P(x_4 = 1 \mid H) \times P(x_5 = 0 \mid H) \times P(x_6 = 2 \mid H) \\ &\quad \times P(x_7 = 3 \mid H) \times P(x_8 = 2 \mid H) \times P(x_9 = 2 \mid H). \end{aligned}$$

We can extract the signature vector of probabilities as follows:

```
x = c(1,1,3,1,0,2,3,2,2) + 1
y = c(1,2,3,4,5,6,7,8,9)
```

For the benign class, we have:

```
p.Benign[as.matrix(data.frame(x,y))]
```

```
[1] 0.33424658 0.83561644 0.08219178 0.82739726
[5] 1.00000000 0.04657534 0.26575342
[8] 0.05753425 0.01369863
```

```
(l.Benign = prod(p.Benign[as.matrix(data.frame(x,y))]))
```

```
[1] 1.852912e-07
```

For the malignant class, we have:

```
p.Malignant[as.matrix(data.frame(x,y))]
```

```
[1] 0.01546392 0.01030928 0.09793814 0.14948454
[5] 1.00000000 0.02577320 0.15463918
[8] 0.03092784 0.10824742
```

```
(l.Malignant = prod(p.Malignant[as.matrix(data.frame(x,y))]))
```

```
[1] 3.114231e-11
```

Based on the multinomial probabilities given in `p.Benign` and `p.Malignant`, the (naïve) likelihood of the undiagnosed case being a benign tumor would thus be 1.86×10^{-7}, while the likelihood of it being a malignant tumor would be 3.11×10^{-11}.[56]

56: **Careful!** These are the likelihoods, not the posteriors.

6. **Computation of posterior:** multiplying the corresponding prior probabilities and likelihoods, we get a quantity that is proportional to the respective posterior probabilities:

$$P(H \mid \mathbf{x}) \propto P(H) \times P(\mathbf{x} \mid H) \approx P(H) \times \prod_{j=1}^{9} P(x_j \mid H).$$

The "likelihoods" can be computed as follows:

```
test.Benign = test.Malignant = NULL
likelihood.Benign = likelihood.Malignant = NULL

for (i in 1:nrow(dat.BC.test)){
  location = rapply(dat.BC.test[i,-10]+1,c)
  for(j in 1:length(location)){
    test.Benign[j] = p.Benign[location[j],j]
    test.Malignant[j] = p.Malignant[location[j],j]
  }

  likelihood.Benign.i = prod(test.Benign)
  likelihood.Malignant.i = prod(test.Malignant)

  likelihood.Benign = c(likelihood.Benign,
                        likelihood.Benign.i)
  likelihood.Malignant = c(likelihood.Malignant,
                           likelihood.Malignant.i)
}
```

The "posteriors" can then be computed as follows:

```
posteriors=cbind(likelihood.Benign*prior.Benign,
                 likelihood.Malignant*prior.Malignant)
```

For the undiagnosed case $\mathbf{x} = (1, 1, 3, 1, 0, 2, 3, 2, 2)$, we obtain:

```
l.Benign*prior.Benign
l.Malignant*prior.Malignant
```

```
[1] 1.209862e-07
[1] 1.080789e-11
```

Comparing the posteriors

$$P(\text{Malignant} \mid \mathbf{x}) < P(\text{Benign} \mid \mathbf{x}),$$

we conclude that the tumor in the undiagnosed case is **likely benign**.[57] We can complete the procedure for all observations in the test set:

57: Note that we have no measurement on how much more likely it is to be benign than to be malignant – the classifier is **not calibrated**.

```
n.test=nrow(dat.BC.test)
prediction=rep(NA, n.test)
prediction[which(posteriors[,1]>posteriors[,2])]="Benign"
prediction[which(posteriors[,1]<posteriors[,2])]="Malignant"
prediction=as.factor(prediction)
table(prediction)
```

```
prediction
   Benign Malignant
       85        55
```

Since we actually know the true outcome for the test subjects, we can take a look at the NBC's performance on the data.

```
table(dat.BC.test$Class,prediction)
```

NBC		prediction	
		Benign	Malignant
actual	Benign	85	8
	Malignant	0	47

Let's take a look at cases where NBC misclassified, and their corresponding posteriors:

```
dat.misclassified = dat.BC.test[
            which(dat.BC.test$Class!=prediction),]
missed.class = prediction[
            which(dat.BC.test$Class!=prediction)]
wrong.classifications = cbind(posteriors[
            which(dat.BC.test$Class!=prediction),])
colnames(wrong.classifications) =
            c("Posterior.Benign","Posterior.Malignant")
wrong.classifications =
            cbind(dat.misclassified, wrong.classifications)
table(wrong.classifications)
```

The confusion matrix tells us that 8 out of 140 cases (5.7%) are misclassified. A closer look at misclassified cases reveals that 3 of the 8 false positives are a result of a posterior probability being 0 (a score level that was not observed in the training set). Taking a close look at ID 130, for instance, all but `Single_Epithelial_Cell_Size`

	Clump_Thickness	Uniformity_of_Cell_Size	Uniformity_of_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Size	Bare_Nuclei	Bland_Chromatin	Normal_Nucleoli	Mitoses	Class	Posterior.Benign	Posterior.Malignant
9	2	1	1	1	2	1	1	1	5	Benign	0	0
130	1	1	1	1	10	1	1	1	1	Benign	0	0
197	8	4	4	5	4	7	7	8	2	Benign	0	0
233	8	4	6	3	3	1	4	3	1	Benign	0	0
253	6	3	3	5	3	10	3	5	3	Benign	0	0
320	4	4	4	4	6	5	7	3	1	Benign	0	0
353	3	4	5	3	7	3	4	6	1	Benign	0	0
658	5	4	5	1	8	1	3	6	1	Benign	0	0

have scores of 1, strongly indicating that tumor is likely benign (perhaps the 10 is a typo?).

Can we prevent misclassification similar to case in ID 130? One way to do so is to replace all 0 probabilities in the likelihood matrices by a small ε (obtained by multiplying a base probability with the smallest non-zero probability), and by re-scaling the columns so that they all add up to 1 (excluding the missing values from the process). If ε is small enough, the larger probabilities will not be affected, in practice.[58]

58: Using a base probability of 10^{-8}%, for instance, would reduce the misclassification rate on the test data to 6/140 (4.3%).

Notes and Comments In practice, various prior distributions or conditional distributions (for the features) can be used; domain matter expertise can come in handy during these steps:

- the naive assumption is made out of **convenience**, as it renders the computation of the likelihood much simpler;
- the variables in a dataset are not typically independent of one another, but NBC still works well with test data (usually) – the method **seems to be robust** against departure from the independence assumption;
- dependency among variables may change the true posterior values, but the class with maximum posterior probabilities is often left **unchanged**;
- in the classification context, we typically get more insight from independent/correlated data than from correlated data;
- NBC works best for independent cases, but optimality can also be reached when dependency among variables inconsistently support one class over another;
- the **choice of a prior** may have a great effect on the classification predictions, as can the presence of outlying observations, especially when |Tr| is small);
- it is not practical to conduct NBC manually, as we have done in this section – a complete implementation can be called by using the method `naiveBayes()` from the `R` package `e1071` (make sure to read the documentation first!), and
- a final reminder that, like the SVM models, NBC is **not calibrated** and should not be used to estimate probabilities.

21.5 Ensemble Learning

In practice, individual learners are often **weak** – they perform better than random guessing would, but not necessarily that much better, or

sufficiently so for specific analytical purposes. In the late 80's, Kearns and Valiant asked the following question: can a set of weak learners be used to create a strong learner? The answer, as it turn out, is yes – *via* **ensemble learning** methods.

As an example, scientists trained 16 pigeons (weak learners, one would assume) to identify pictures of magnified biopsies of possible breast cancers. On average, each pigeon had an accuracy of about 85%, but when the most popular answer among the group was selected, the accuracy jumped to 99% [21].[59]

59: The material of this section closely follows [18].

21.5.1 Bagging

Bootstrap aggregation (also known as **bagging**) is an extension of bootstrapping. Originally, bootstrapping was used in situations where it is nearly impossible to compute the variance of a quantity of interest by exact means (see Section 20.3.2).

But it can also be used to improve the performance of various statistical learners, especially those that exhibit **high variance** (such as CART).[60] Given a learning method, bagging can be used to **reduce the variance** of that method.

60: Low variance methods, in comparison, are those for which the results, structure, predictions, etc. remain roughly similar when using different training sets, such as OLS when $N/p \gg 1$, and are less likely to benefit from the use of ensemble learning.

If $Z_1, \dots, Z_B$ are **independent** predictions at $\mathbf{x} \in \text{Te}$, say, with

$$\text{Cov}(Z_i, Z_j) = \begin{cases} \sigma^2 & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

the central limit theorem states that

$$\begin{aligned} \text{Var}(\overline{Z}) = \text{Var}\left(\frac{Z_1 + \cdots + Z_B}{B}\right) &= \frac{1}{B^2}\text{Var}(Z_1 + \cdots + Z_B) \\ = \frac{1}{B^2}\sum_{i,j=1}^{B}\text{Cov}(Z_i, Z_j) &= \frac{1}{B^2}\sum_{k=1}^{B}\text{Var}(Z_i) = \frac{\sigma^2}{B}. \end{aligned}$$

In other words, averaging a set of observations reduces the variance as $\sigma^2 \geq \frac{\sigma^2}{B}$ for all $B \in \mathbb{N}$. In practice, this conclusion seems, at first, not to be as interesting as originally intended since we do not usually have access to multiple training sets. However, resampling methods can be used to generate multiple **bagging training sets** from the original training set Tr.

Let $B > 1$ be an integer. We generate B bootstrapped training sets from Tr by sampling $N = |\text{Tr}|$ observations from Tr, with replacement, to yield

$$\text{Tr}_1, \dots, \text{Tr}_B,$$

and train a model $\hat{f}_i$ (for regression) or $\hat{C}_i$ (for classification) on each Tr_i, $i = 1, \dots, B$; for each $\mathbf{x}^* \in \text{Te}$, we then have B predictions

$$\begin{aligned} &\hat{f}_1(\mathbf{x}^*), \dots, \hat{f}_B(\mathbf{x}^*) \quad \text{(for regression)} \\ &\hat{C}_1(\mathbf{x}^*), \dots, \hat{C}_B(\mathbf{x}^*) \quad \text{(for classification).} \end{aligned}$$

The **bagging prediction** at $\mathbf{x}^* \in \text{Te}$ is the average of all predictions

$$\hat{f}_{\text{Bag}}(\mathbf{x}^*) = \frac{1}{B}\sum_{i=1}^{B}\hat{f}_i(\mathbf{x}^*) \quad \text{(for regression)},$$

or the most frequent prediction

$$\hat{C}_{\text{Bag}}(\mathbf{x}^*) = \text{Mode}\{\hat{C}_1(\mathbf{x}^*), \ldots, \hat{C}_B(\mathbf{x}^*)\} \quad \text{(for classification)}.$$

Bagging is particularly helpful in the CART framework; to take full advantage of bagging, however, the trees should be grown **deep** (i.e., no pruning), as their complexity will lead to high variance but low bias (thanks to the bias-variance trade-off).

In practice, the bagged tree predictions would also have low bias, but the variance will be reduced by the bagging process; bagging with 100s/1000s of trees typically produces greatly improved predictions (at the cost of interpretability, however).

Out-of-Bag Error Estimation As is usually the case with supervised models, we will need to estimate the test error for a bagged model. There is an easy way to provide the estimate without relying on cross-validation, which is computationally expensive when N is large.

The jth model is fit to the bootstrapped training set Tr_j, $j = 1, \ldots, B$. We can show that, on average, each of the Tr_j contains $\approx 2/3$ distinct observations of Tr, which means that $\approx 1/3$ of the training observations are not used to build the model (we refer to those observations are **out-of-bag (OOB) observations**).

We can then predict the response y_i for the ith observation in Tr using only those models for which $\mathbf{x}_i$ was OOB; there should be about $B/3$ such predictions, and

$$\hat{y}_i = \text{Avg}\{\hat{f}_j(\mathbf{x}_i) \mid \mathbf{x}_i \in \text{OOB}(\text{Tr}_j) = \text{Tr} \setminus \text{Tr}_j\} \quad \text{(for regression)}$$
$$\hat{y}_i = \text{Mode}\{\hat{C}_j(\mathbf{x}_i) \mid \mathbf{x}_i \in \text{OOB}(\text{Tr}_j)\} \quad \text{(for classification)}.$$

The OOB MSE (or the OOB misclassification rate) are thus good Te error estimates since none of the predictions are given by models that used the test observations in their training.

Variable Importance Measure Bagging improves the accuracy of stand-alone models, but such an improvement comes at the cost of **reduced interpretability**, especially in the case of CART: the bagged tree predictions cannot, in general, be expressed with the help of a single tree. In such a tree, the relative importance of the features is linked to the **hierarchy of splits**.[61]

61: Namely, the most "important" variables appear in the earlier splits.

For bagged **regression trees**, a measure such as the total amount in decreased SSE due to splits over a given predictor, in which we compare SSE in trees with these splits against SSE in trees without these splits, averaged over the B bagged trees, provides a summary of the importance of each variable (large scores indicate important variables). For bagged **classification trees**, we would replace SSE with the **Gini index**, instead.

Another approach might be to weigh the importance of a factor **inversely proportionally** to the level in which it appears (if at all) in each bagging tree and to average over all bagging trees.

For instance, if predictor X_1 appears in the 1st split level of bagged tree 1, the 4th split level of bagged tree 2, and the 3rd split level of bagged tree 5, whereas predictor X_2 appears in the 2nd, 2nd, 3rd, and 5th split levels of bagged trees 2, 3, 4, 5 (resp.), then the relative importance of each predictor over the 5 bagged trees is

$$X_1 : (1 + 1/4 + 0 + 0 + 1/3) \cdot 1/5 = 19/60 = 0.32$$
$$X_2 : (0 + 1/2 + 1/2 + 1/3 + 1/5) \cdot 1/5 = 23/75 = 0.31;$$

the first variable is nominally more important than the second.

Example We once again revisit the Iowa Housing Price example of Sections 20.5.2 and 21.4.1. Recall that we had built a training set `dat.train` with $n = 1160$ observations relating to the selling price `SalePrice` of houses in Ames, Iowa.

```
dat.Housing = read.csv("VE_Housing.csv", header=TRUE,
                       stringsAsFactors = TRUE)
missing = attributes(which(apply(is.na(dat.Housing), 2,
                                 sum)>0))$names
dat.Housing.new = dat.Housing[,!colnames(dat.Housing)
                               %in% missing]
dat.Housing.new = subset(dat.Housing.new, select = -c(Id))
set.seed(1234) # for replicability
n.train = 1160
ind.train = sample(1:nrow(dat.Housing.new), n.train)
dat.train = dat.Housing.new[ind.train,]
dat.test = dat.Housing.new[-ind.train,]
```

We build a regression tree bagging model using the `R` package `ipred`, with 150 bags, and using an OOB error estimate.

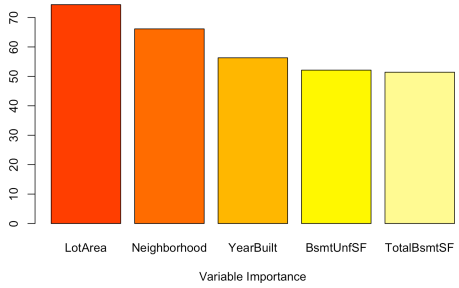
```
set.seed(1234)
library(ipred)
(bag <- bagging(formula = SalePrice ~ ., data = dat.train,
            nbagg = 150, coob = TRUE, control =
            rpart::rpart.control(minsplit = 5, cp = 0),
            importance = TRUE))
```

```
Bagging regression trees with 150 bootstrap replications
Out-of-bag estimate of root mean squared error:  29.2526
```

We can display the 5 most important variables:

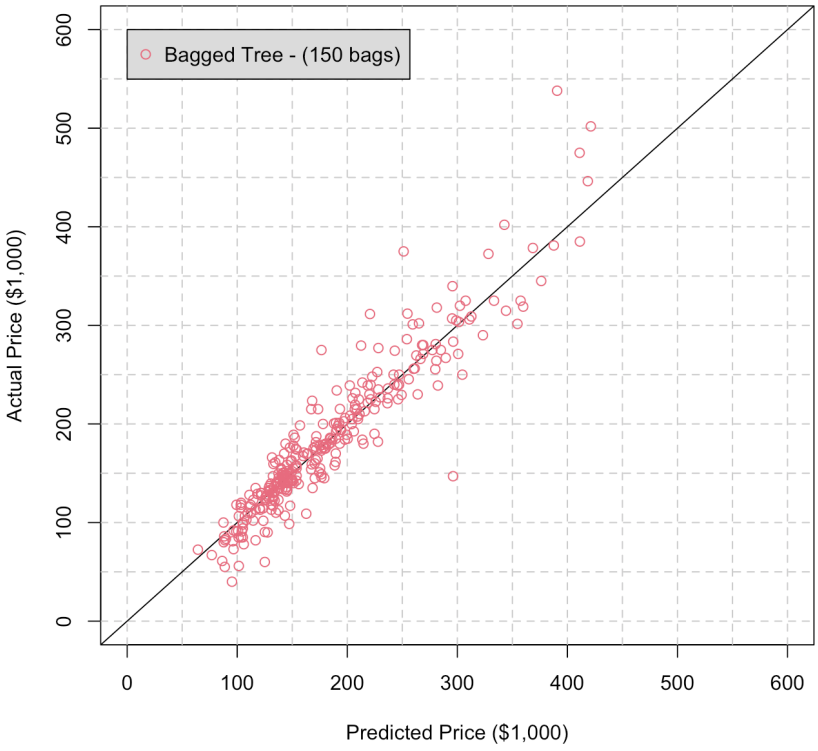
```
p=ncol(dat.train)-1
vim <- data.frame(var=names(dat.train[,1:p]),
                  imp=caret::varImp(bag))
```

```
vim.plot <- vim[order(vim$Overall, decreasing=TRUE),]
vim.plot <- vim.plot[1:5,]
barplot(vim.plot$Overall, names.arg=rownames(vim.plot),
        col=heat.colors(5), xlab='Variable Importance')
```



The predictions on the testing data are obtained (and plotted) as follows:

```
yhat.bag = predict(bag, newdata=dat.test)

xlimit = ylimit = c(0,600)
plot(NA, col=2, xlim=xlimit, ylim=ylimit,
     xlab="Predicted Price ($1,000)",
     ylab="Actual Price ($1,000)")
abline(h=seq(0,600, length.out=13), lty=2, col="grey",
       v=seq(0,600, length.out=13))
abline(a=0, b=1)
points(yhat.bag, dat.test$SalePrice, col=2)
legend(0,600, legend=c("Bagged Tree - (150 bags)"),
       col=c(2), pch=rep(1), bg='light grey')
```



The correlation measure between predicted and actual sale prices on the `dat.test` is:

```
cor(yhat.bag, dat.test$SalePrice)
```

```
[1] 0.9413044
```

How does that compare to the previous MARS and CART models?

21.5.2 Random Forests

In a bagging procedure, we fit models on various training sets, and we use the central limit theorem, assuming independence of the models, to reduce the variance.

In practice, however, the independence assumption is rarely met: if there are only a few strong predictors in Tr, each of the bagged models (built on the bootstrapped training sets Tr_i) is likely to be similar to the others, and the various bagging predictions are unlikely to be un-correlated, so that

$$\text{Var}(\hat{y}^*_{\text{Bag}}) \neq \frac{\sigma^2}{B}, \quad \mathbf{x}^* \in \text{Te};$$

averaging highly correlated quantities does not reduce the variance significantly.[62]

62: The central limit theorem assumption of independence of observations is necessary.

With a small tweak, however, we can **decorrelate the bagged models**, leading to variance reduction when we average the (bagged) predictions. **Random forests** also build models on B bootstrapped training samples, but each model is built out of a **random subset** of predictors.

For decision trees, every time a split is considered, the set of allowable predictors is selected from a random subset of m predictors out of the full p predictors. By selecting predictors randomly for each model, we lose out on building the best possible model on each training sample, but we also reduce the chance of them being correlated. For a test observation $\mathbf{x}^*$, the B predictions are combined as in bagging to yield the **random forest prediction**.

If $m = p$, random forests reduce to bagged models; in practice we use $m \approx \sqrt{p}$ for classification and $m \approx p/3$ for regression. When the predictors are highly correlated, however, smaller values of m are recommended.

Example We revisit the Wine example of Section 21.4.3, using the `R` package `randomForest`.

```
wine = read.csv("wine.csv", header=TRUE,
                stringsAsFactors = TRUE)
wine$Class = as.factor(wine$Class)
```

Let's implement a 70%/30% training/testing split:

```
set.seed(1111)
ind.train <- sample(nrow(wine), 0.8*nrow(wine),
                    replace = FALSE)
dat.train <- wine[ind.train,]
dat.test <- wine[-ind.train,]
```

There are $p = 13$ predictors in the dataset, so we should use $m \approx \sqrt{13} \approx 4$ predictors at each split. Keep in mind, however, that we have seen that some of the variables are correlated, so we will try models for $m = 1$, $m = 2$, $m = 3$, and $m = 4$.

```
wine.rf.1 <- randomForest::randomForest(Class ~ .,
    data = dat.train, ntree = 500, mtry = 1,
    importance = TRUE)
wine.rf.1
```

```
No. of variables tried at each split: 1
        OOB estimate of  error rate: 0.7%
Confusion matrix:
   1  2  3 class.error
1 47  0  0  0.00000000
2  0 54  1  0.01818182
3  0  0 40  0.00000000
```

```
wine.rf.2 <- randomForest::randomForest(Class ~ .,
    data = dat.train, ntree = 500, mtry = 2,
    importance = TRUE)
wine.rf.2
```

```
No. of variables tried at each split: 2
        OOB estimate of  error rate: 0.7%
Confusion matrix:
   1  2  3 class.error
1 47  0  0  0.00000000
2  0 54  1  0.01818182
3  0  0 40  0.00000000
```

```
wine.rf.3 <- randomForest::randomForest(Class ~ .,
    data = dat.train, ntree = 500, mtry = 3,
    importance = TRUE)
wine.rf.3
```

```
No. of variables tried at each split: 3
        OOB estimate of  error rate: 1.41%
Confusion matrix:
   1  2  3 class.error
1 46  1  0  0.02127660
2  0 54  1  0.01818182
3  0  0 40  0.00000000
```

```
wine.rf.4 <- randomForest::randomForest(Class ~ .,
    data = dat.train, ntree = 500, mtry = 4,
    importance = TRUE)
wine.rf.4
```

```
No. of variables tried at each split: 4
        OOB estimate of  error rate: 1.41%
Confusion matrix:
   1  2  3 class.error
1 46  1  0  0.02127660
2  0 54  1  0.01818182
3  0  0 40  0.00000000
```

In this example, then, the choice of m only introduces slight differences. Obviously, this will not always be the case.

21.5.3 Boosting

Another general approach to improving prediction results for statistical learners involves creating a sequence of models, each improving over the previous model in the series. **Boosting** does not involve bootstrap sampling; instead, it fits models on a hierarchical sequence of **residuals**, but it does so in a **slow manner**.

For regression problems, we proceed as follows:

1. set $\hat{f}(\mathbf{x}) = 0$ and $r_i = y_i$ for all $\mathbf{x}_i \in \text{Tr}$;
2. for $b = 1, 2, \ldots, B$:
   a) fit a model $\hat{f}^b$ to the training set $(\mathbf{X}, \mathbf{r})$;
   b) update the regression function $\hat{f} := \hat{f} + \lambda \hat{f}^b$;
   c) update the residuals $r_i := r_i - \lambda \hat{f}^b(\mathbf{x}_i)$ for all $\mathbf{x}_i \in \text{Tr}$;
3. output the boosted model $\hat{f}(\mathbf{x}) = \lambda(\hat{f}^1(\mathbf{x}) + \cdots + \hat{f}^B(\mathbf{x}))$.

In this version of the algorithm, boosting requires three tuning parameters:

- the **number of models** B, which can be selected through cross-validation (boosting can overfit if B is too large);
- the **shrinkage parameter** λ (typically, $0 < \lambda \ll 1$), which controls the **boosting learning rate** (a small λ needs a large B, in general); the optimal λ and B can be found *via* cross-validation, and
- although not explicitly stated, we also need the learning models to reach some **complexity threshold**.

Variants of the boosting algorithm allowing for classification and for varying weights depending on performance regions in predictor space also exist and are quite popular. While the various *No Free Lunch* theorems guarantee that no supervised learning algorithm is always best regardless of context/data, the combination of AdaBoost with weak CART learners is seen by many as the best "out-of-the-box" classifier.

Example Consider the Credit.csv dataset [18]; the task is to determine the credit card balance based on a number of other factors.

```
str(Credit)
```

```
'data.frame':   400 obs. of  12 variables:
 $ X         : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Income    : num  14.9 106 104.6 148.9 55.9 ...
 $ Limit     : int  3606 6645 7075 9504 4897 8047 3388 7114 3300 6819 ...
 $ Rating    : int  283 483 514 681 357 569 259 512 266 491 ...
 $ Cards     : int  2 3 4 3 2 4 2 2 5 3 ...
 $ Age       : int  34 82 71 36 68 77 37 87 66 41 ...
 $ Education: int  11 15 11 11 16 10 12 9 13 19 ...
 $ Gender    : Factor w/ 2 levels "Female","Male": 2 1 2 1 2 2 1 2 1 1 ...
 $ Student   : Factor w/ 2 levels "No","Yes": 1 2 1 1 1 1 1 1 1 2 ...
 $ Married   : Factor w/ 2 levels "No","Yes": 2 2 1 1 2 1 1 1 1 2 ...
 $ Ethnicity: Factor w/ 3 levels "African American",..: 3 2 2 2 3 3 1 2 3 1 ...
 $ Balance   : int  333 903 580 964 331 1151 203 872 279 1350 ...
```

We remove the index variable, and create binary variables for all categorical levels in the data.

```
Credit <- Credit[,-c(1)]
Credit$Gender.dummy <- ifelse(
        Credit$Gender == "Female",1,0)
Credit$Student.dummy <- ifelse(
        Credit$Student == "Yes",1,0)
Credit$Married.dummy <- ifelse(
        Credit$Married == "Yes",1,0)
Credit$Ethnicity.AA.dummy <- ifelse(
        Credit$Ethnicity == "African American",1,0)
Credit$Ethnicity.A.dummy <- ifelse(
        Credit$Ethnicity == "Asian",1,0)
```

The dataset under consideration will then have the following shape:

```
Credit <- Credit[,c(1:6,12:16,11)]
str(Credit)
```

```
'data.frame':   400 obs. of  12 variables:
 $ Income            : num  14.9 106 104.6 148.9 55.9 ...
 $ Limit             : int  3606 6645 7075 9504 4897 8047 3388 7114 3300 6819 ...
 $ Rating            : int  283 483 514 681 357 569 259 512 266 491 ...
 $ Cards             : int  2 3 4 3 2 4 2 2 5 3 ...
 $ Age               : int  34 82 71 36 68 77 37 87 66 41 ...
 $ Education         : int  11 15 11 11 16 10 12 9 13 19 ...
 $ Gender.dummy      : num  0 1 0 1 0 0 1 0 1 1 ...
 $ Student.dummy     : num  0 1 0 0 0 0 0 0 0 1 ...
 $ Married.dummy     : num  1 1 0 0 1 0 0 0 0 1 ...
 $ Ethnicity.AA.dummy: num  0 0 0 0 0 0 1 0 0 1 ...
 $ Ethnicity.A.dummy : num  0 1 1 1 0 0 0 1 0 0 ...
 $ Balance           : int  333 903 580 964 331 1151 203 872 279 1350 ...
```

We pick 300 of the 400 observations to be part of the training set:

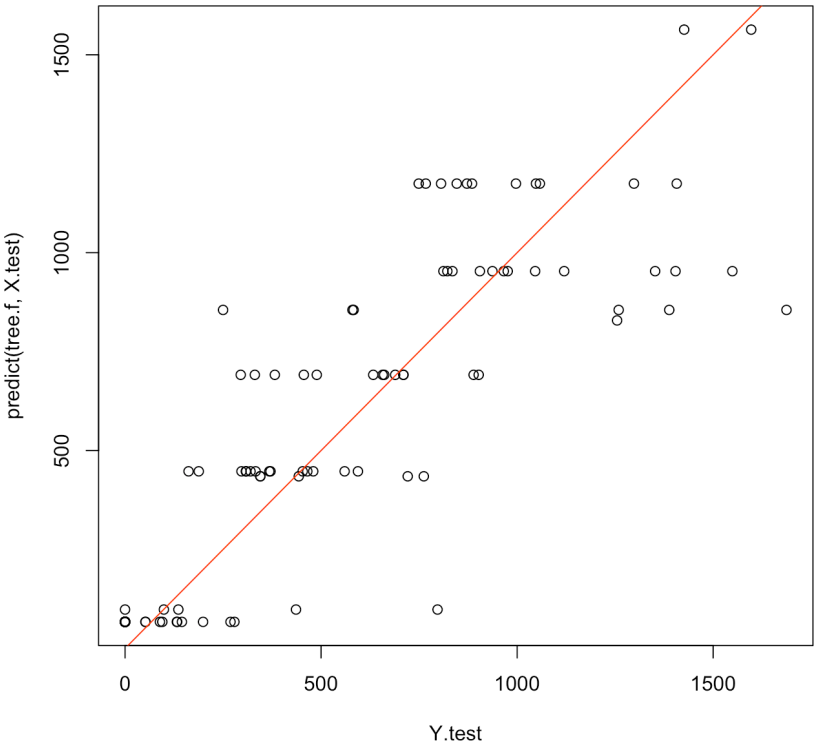
```
set.seed(1234)
ind = sample(1:nrow(Credit), size = 300)
Credit.train = Credit[ind,]
Credit.test = Credit[-ind,]
```

We use $\lambda = 0.005$ as a shrinkage parameter and $B = 2000$ models.

```
lambda = 0.005
X <- Credit.train[,1:11] # predictors
r <- Credit.train[,12]   # response
X.test <- Credit.test[,1:11]
Y.test = Credit.test[,12]
```

We start by building the first iteration of the boosting model, using R's `tree` package, and look at its predictions on the test data:

```
tree.f <- tree::tree(r ~ ., data = data.frame(cbind(X,r)),
                     na.action=na.pass)
r <- r - lambda * predict(tree.f, X)
plot(Y.test,predict(tree.f, X.test))
abline(0,1,col="red")
```



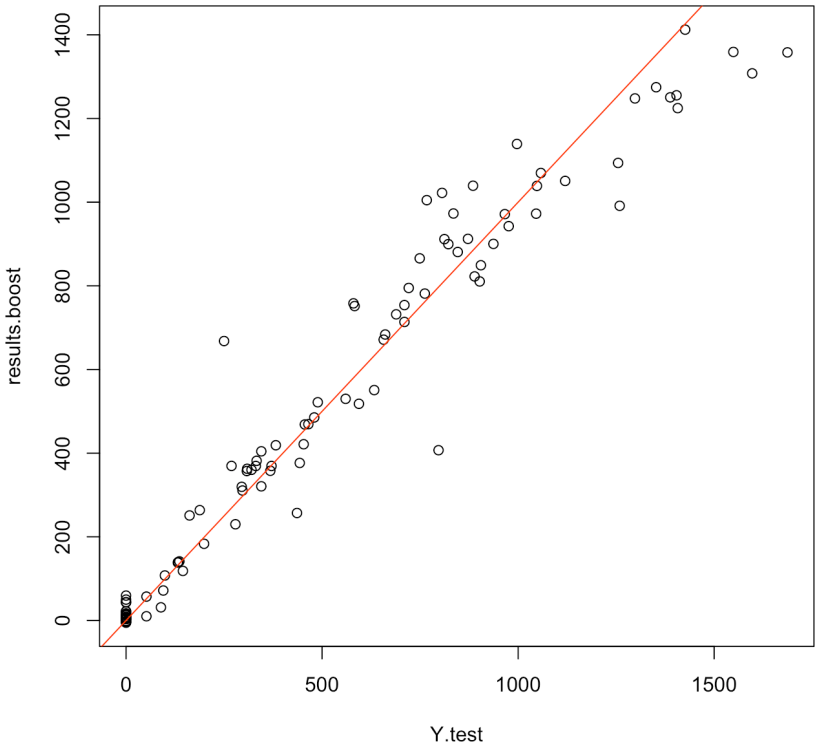
Visually, the performance seems middling, which is born by the correlation metric between the actual test observations and the predicted test observations.

```
cor(Y.test,predict(tree.f, X.test))
```

```
[1] 0.8640351
```

Let us compare with the results of boosting with 2000 models.

```
results.boost <- 0*Y.test
B=2000
tree.full <- c()
tree.snipped <- c()

for(b in 1:B){
  tree.full[[b]] <- tree::tree(r ~ .,
      data = data.frame(cbind(X,r)), na.action=na.pass)
  tree.snipped[[b]] <- tree::tree(r ~ .,
      data = data.frame(cbind(X,r)), na.action=na.pass)
  r <- r - lambda * predict(tree.snipped[[b]], X)
  results.boost = results.boost + lambda *
      predict(tree.snipped[[b]], X.test)
}

plot(Y.test,results.boost)
abline(0,1,col="red")
```



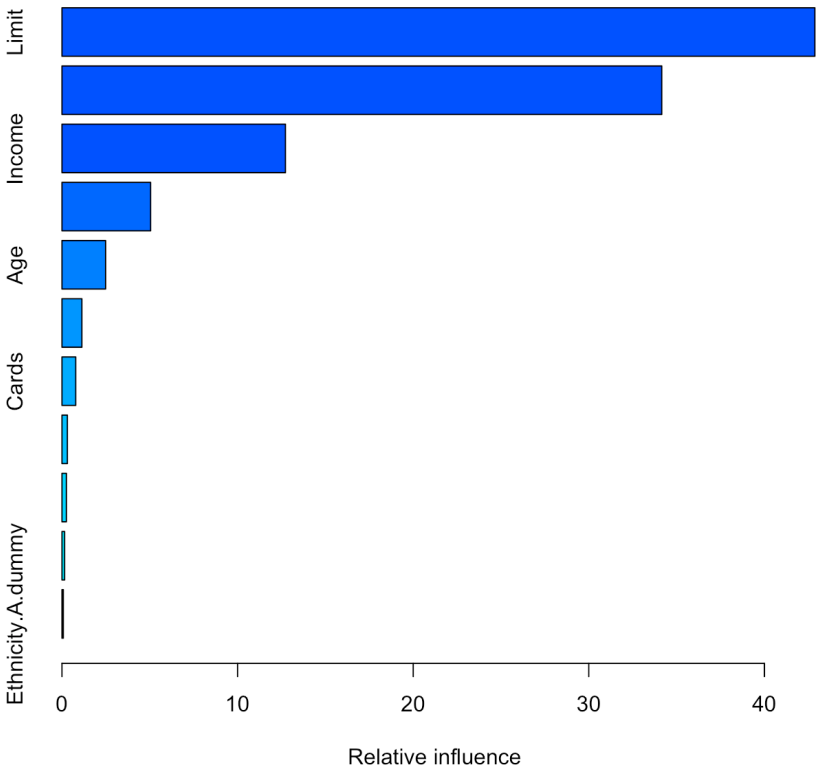
Visually, the performance is much improved; the correlation metric also agrees:

```
cor(Y.test,results.boost)
```

```
[1] 0.9734103
```

We can also use the pre-built `gbm` package to achieve sensibly the same results, and get the influential predictors as a bonus:

```
boost.credit = gbm::gbm(Balance~., data = Credit.train,
            distribution = "gaussian", n.trees = 10000,
            shrinkage = 0.01, interaction.depth = 4)
```




```
summary(boost.credit)
```

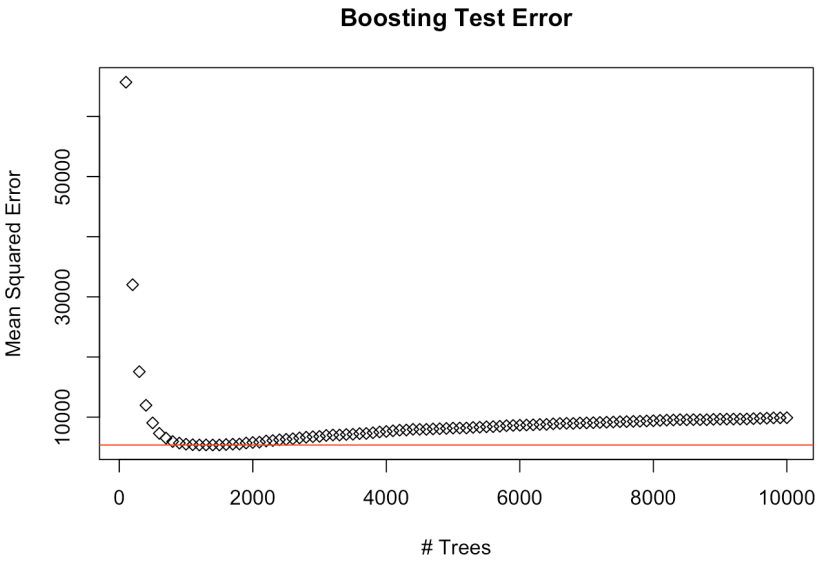
```
                     var     rel.inf
Limit              Limit 42.87367610
Rating            Rating 34.15766697
Income            Income 12.72692829
Student.dummy Student.dummy  5.04913001
Age                  Age  2.48572012
Education      Education  1.13432102
Cards              Cards  0.78545388
Ethnicity.AA.dummy Ethnicity.AA.dummy  0.30692238
Married.dummy Married.dummy  0.25299822
Gender.dummy   Gender.dummy  0.15135520
Ethnicity.A.dummy Ethnicity.A.dummy  0.07582781
```

Not surprisingly, ethnicity and gender have very little influence on the model.

In order to determine the optimal number of models B to use,[63] we seek to minimize the prediction MSE:

63: Is it really necessary to run 10,000 models?

```
n.trees = seq(from = 100, to = 10000, by = 100)
predmat = predict(boost.credit, newdata = Credit.test,
                  n.trees = n.trees)
boost.err = with(Credit.test,
              apply( (predmat - Balance)^2, 2, mean) )
plot(n.trees, boost.err, pch = 23, xlab = "# Trees",
     ylab = "MSE", main = "Boosting Test Error")
abline(h = min(boost.err), col = "red")
```

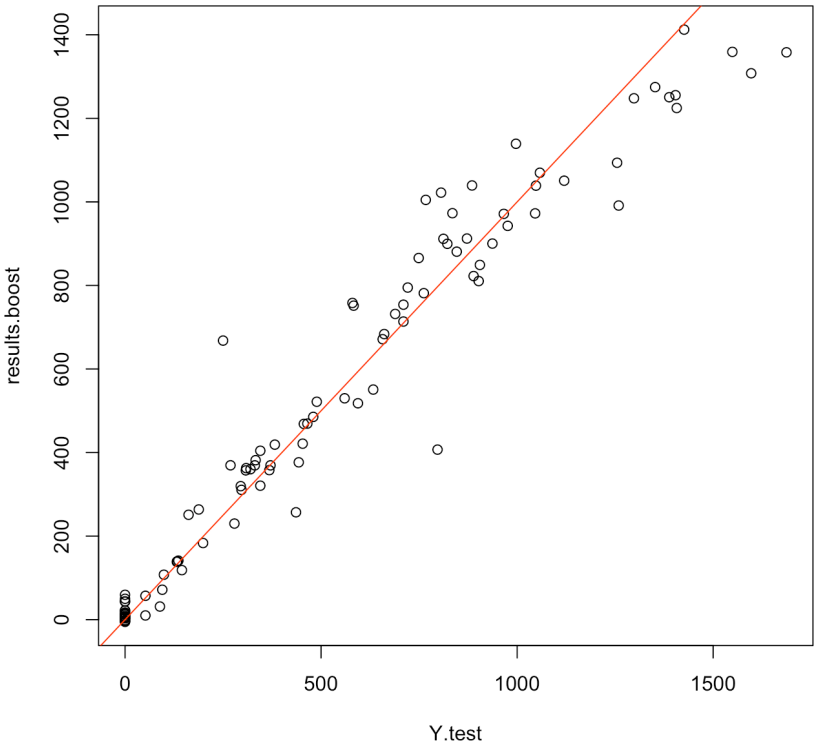



```
which(boost.err==min(boost.err))
```

```
1200
  12
```

The optimal `gbm` boosted model (with parameters as in the `gbm()` call above) is thus:

```
results.boost.gbm = predict(boost.credit,
        newdata = Credit.test, n.trees = 1200)
plot(Y.test,results.boost)
abline(0,1,col="red")
```




```
cor(Y.test,results.boost)
```

```
[1] 0.9734103
```

AdaBoost **Adaptive Boosting** (AdaBoost) adapts dynamic boosting to a set of models in order to minimize the error made by the individual weak models.[64] The "adaptive" part means that any new weak learner that is added to the boosted model is **modified to improve the predictive power** on instances which were "mis-predicted" by the (previous) boosted model.

64: Such as "stubby" trees or "coarse" linear models.

The main idea behind **dynamic boosting** is to consider a **weighted sum** of weak learners whose weights are adjusted so that the prediction error is minimized.

Consider a binary classification context, where

$$\text{Tr} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, N\}, \quad \text{and } y_i \in \{-1, +1\} \quad \forall i = 1, \ldots, N.$$

The **boosted classifier** is a function

$$F(\mathbf{x}) = \sum_{b=1}^{B} c_b f_b(\mathbf{x}),$$

where $f_b(\mathbf{x}) \in \{-1, +1\}$ and $c_b \in \mathbb{R}$ for all b and all $\mathbf{x}$. The **class prediction** at $\mathbf{x}$ is simply $\text{sign}(F(\mathbf{x}))$.

The **AdaBoost contribution** comes in at the modeling stage, where, for each $\mu \in \{1, \ldots, B\}$ the weak learner f_μ is trained on a weighted version

of the original training data, with observations that are misclassified by the partial boosted model

$$F_\mu(\mathbf{x}) = \sum_{b=1}^{\mu-1} c_b f_b(\mathbf{x})$$

given larger weights; AdaBoost estimates the weights w_i, $i = 1, \ldots, N$ at each of the boosting steps $b = 1, \ldots, B$.

Real AdaBoost [10] is a generalization of this approach which does away with the constants c_b:

1. initialize the weights $\mathbf{w}$, with $w_i = 1/N$, for $1 \leq i \leq N$;
2. for $b = 1, \ldots, B$, repeat the following steps:
   a) fit the class probability estimate $p_m(\mathbf{x}) = P(y = 1 \mid \mathbf{x}, \mathbf{w})$, using the weak learner algorithm of interest;
   b) define $f_b(\mathbf{x}) = \frac{1}{2} \log \frac{p_b(\mathbf{x})}{1-p_b(\mathbf{x})}$;
   c) set $w_i \leftarrow w_i \exp\{-y_i f(\mathbf{x}_i)\}$, for $1 \leq i \leq N$;
   d) re-normalize so that $\|\mathbf{w}\|_1 = 1$;
3. output the classifier

$$F(\mathbf{x}) = \text{sign}\left\{\sum_{b=1}^{B} f_b(\mathbf{x})\right\}.$$

For regression tasks, this procedure must be modified to some extent (in particular, the equivalent task of assigning larger weights to currently misclassified observations at a given step is to train the model to predict (shrunken) residuals at a given step... more or less).

Since boosting is susceptible to overfitting (unlike bagging and random forests), the optimal number of boosting steps B should be derived from cross-validation.

Example The Python library `scikit-learn` provides a useful implementation of AdaBoost. In order to use it, a base estimator (that is to say, a week learner) must first be selected.

In what follows, we will use a decision tree classifier. Once this is achieved, a `AdaBoostClassifier` object is created, to which is fed the weak learner, the **number of estimators** and the **learning rate** (a.k.a. the shrinkage parameter, which we have seen is a small positive number).

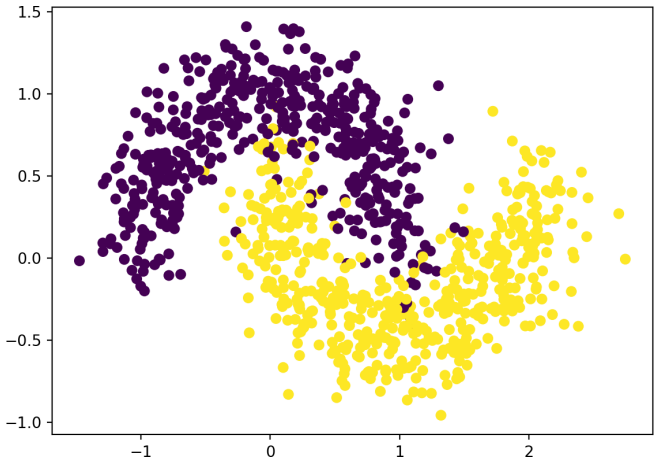
In general, small learning rates require a large number of estimators to provide adequate performance. By default, `scikit-learn`'s implementation uses 50 estimators with a learning rate of 1.

We use the classic *Two-Moons* dataset consisting of two interleaving half circles with added noise, in order to test and compare classification results for AdaBoost (and eventually Gradient Boosting, see below).

This dataset is conveniently built into `scikit-learn` and accessible *via* `make_moons()`, which returns a data matrix `X` and a label vector `y`. We can treat the dataset as a complete training set as we eventually use **cross-validation** to estimate the test error.[65]

65: The AdaBoost code on the Two-Moons dataset was lifted from an online source whose location cannot be recovered at the moment.

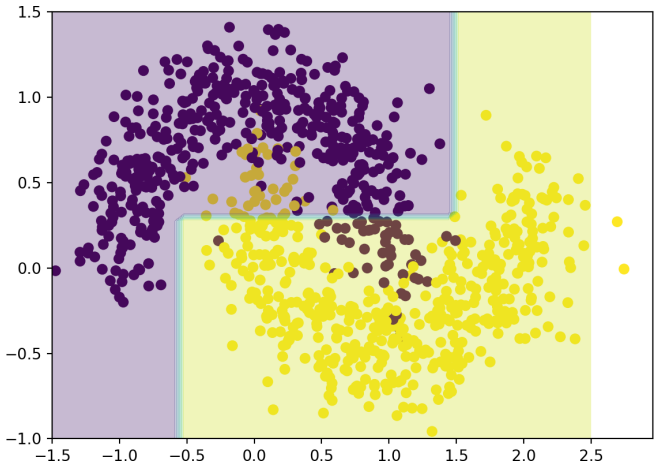
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_moons
N = 1000
X,Y = make_moons(N,noise=0.2)
plt.scatter(X[:,0],X[:,1], c=Y)
plt.show()
```



We first attempt to classify the data using `DecisionTreeClassifier()` with a maximum depth of 3.[66]

66: This same structure will later be used for our weak learners.

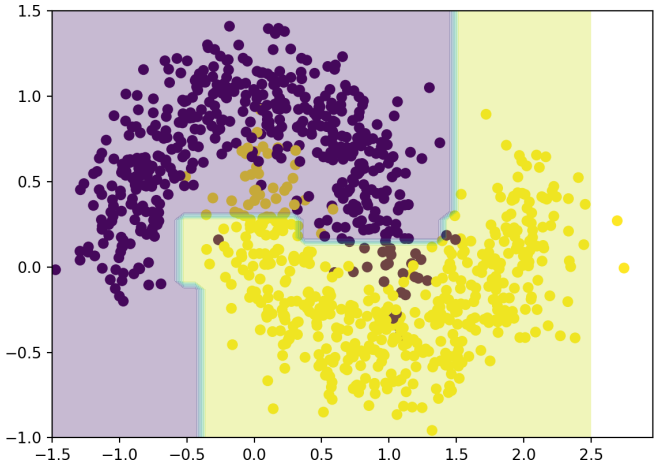
```
clf = DecisionTreeClassifier(max_depth=3)
clf.fit(X,Y)
xx,yy = np.meshgrid(np.linspace(-1.5,2.5,50),
                    np.linspace(-1,1.5,50))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.scatter(X[:,0],X[:,1], c = Y)
plt.contourf(xx,yy,Z,alpha=0.3)
plt.show()
```



As can be seen from the display, this single decision tree does not provide the best of fits.

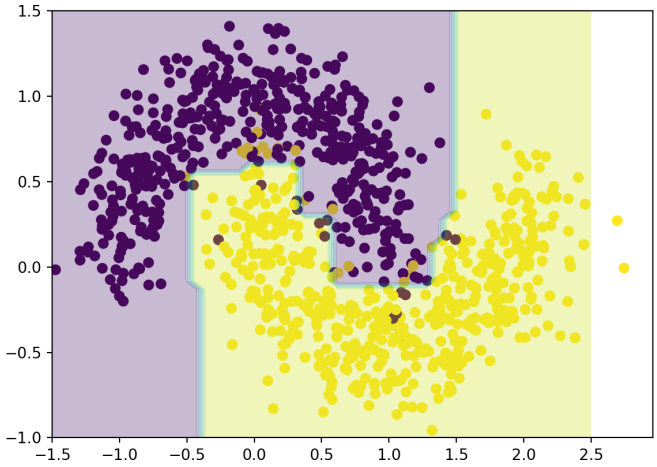
Next, we build an AdaBoost classifier. We first consider a model with $B = 5$ decision trees, and with a learning rate $\lambda = 1/10$.

```
ada = AdaBoostClassifier(clf, n_estimators=5,
                         learning_rate=0.1)
ada.fit(X,Y)
xx,yy = np.meshgrid(np.linspace(-1.5,2.5,50),
                    np.linspace(-1,1.5,50))
Z = ada.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.scatter(X[:,0],X[:,1], c = Y)
plt.contourf(xx,yy,Z,alpha=0.3)
plt.show()
```



Finally, we build an AdaBoost classifier with $B = 10$ decision trees and with a learning rate $\lambda = 1/10$.

```
ada = AdaBoostClassifier(clf, n_estimators=10,
                         learning_rate=0.1)
ada.fit(X,Y)
xx,yy = np.meshgrid(np.linspace(-1.5,2.5,50),
                    np.linspace(-1,1.5,50))
Z = ada.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.scatter(X[:,0],X[:,1], c = Y)
plt.contourf(xx,yy,Z,alpha=0.3)
plt.show()
```



The AdaBoosted tree is better at capturing the dataset's structure. Of course, until we evaluate the performance on an independent test set, this could simply be a consequence of overfitting (one of AdaBoost's main weaknesses, as the procedure is sensitive to outliers and noise). We can guard against this eventuality by adjusting the learning rate (which provides a step size for the algorithm's iterations).

To find the optimal value for the learning rate and the number of estimators, one can use the `GridSearchCV` method from `sklearn.model_selection`, which implements cross-validation on a grid of parameters. It can also be parallelized, in case the efficiency of the algorithm should ever need improving (that is not necessary on such a small dataset, but could prove useful with larger datasets).

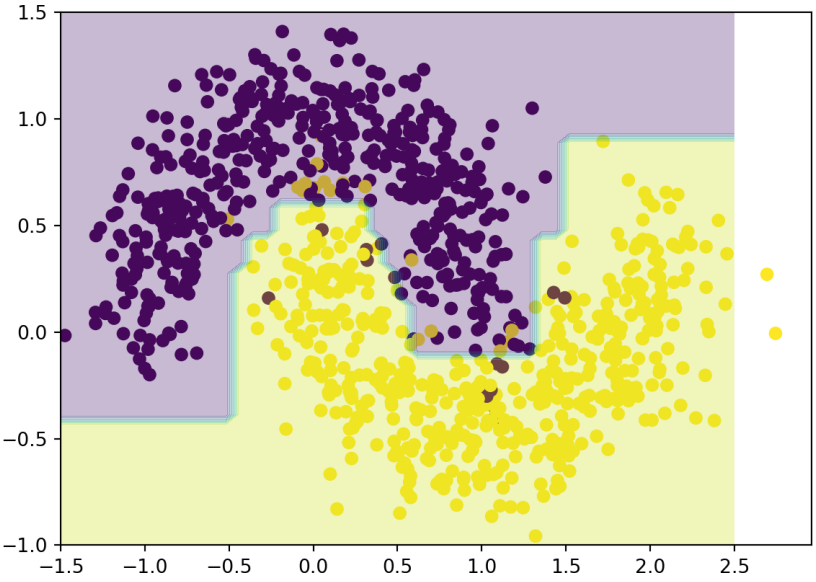
```
import random
random.seed(10)

from sklearn.model_selection import GridSearchCV
params = {
     'n_estimators': np.arange(10,300,10),
     'learning_rate': [0.01, 0.05, 0.1, 1],
 }

grid_cv = GridSearchCV(AdaBoostClassifier(),
            param_grid= params, cv=5, n_jobs=1)
grid_cv.fit(X,Y)
grid_cv.best_params_
```

```
{'learning_rate': 0.05, 'n_estimators': 200}
```

The results show that, given the selected grid search ranges, the optimal parameters (those that provide the best cross-validation fit for the data) are 200 estimators with a learning rate of 0.05 (these parameters change if `cv` and `n_jobs` are modified, and with different random seeds).

The plot of the model with these parameters indeed shows that the fit looks quite acceptable.

```
ada = grid_cv.best_estimator_
xx,yy = np.meshgrid(np.linspace(-1.5,2.5,50),
                    np.linspace(-1,1.5,50))
Z = ada.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.scatter(X[:,0],X[:,1], c = Y)
plt.contourf(xx,yy,Z,alpha=0.3)
plt.show()
```

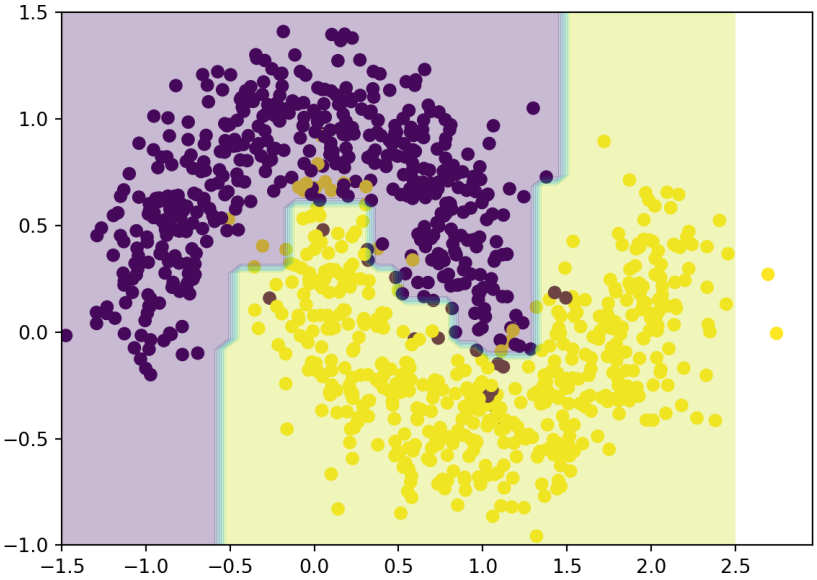


Gradient Boosting The implementation of **gradient boosting** is simpler than that of AdaBoost. The idea is to first fit a model, then to compute the residuals generated by this model. Next, a new model is trained, but on the residuals instead of on the original response. The resulting model is added to the first one. Those steps are repeated a sufficient number of times. The final model will be a sum of the initial model and of the subsequent models trained on the chain of residuals.

We will not go into the nitty-gritty of gradient boosting here (see [20] for details), but we showcase how it would be applied on the Two-Moons dataset.[67]

67: Remember that without an estimate of the test error, we cannot use these classifiers for predictive purposes due to avoid overfitting issues.

```
from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier(n_estimators=9,
                                 learning_rate=0.5)
gbc.fit(X,Y)
xx,yy = np.meshgrid(np.linspace(-1.5,2.5,50),
                    np.linspace(-1,1.5,50))
Z = gbc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.scatter(X[:,0],X[:,1], c = Y)
plt.contourf(xx,yy,Z,alpha=0.3)
plt.show()
```



21.6 Exercises

1. Repeat the vowel classification example on PCA-reduced data.
2. Conduct a pre-analysis exploration as in the Wine example to remove variables in the 2011 Gapminder, the Iowa Housing, and the Vowel datasets before conducting the analyses, as in the examples.
3. Construct and evaluate naïve Bayes classifiers for the Wine and for the 2011 Gapminder dataset.
4. Construct and evaluate CART models for the Wine and for the Wisconsin Breast Cancer datasets.
5. Construct and evaluate ANN models for the 2011 Gapminder, for the Iowa Housing, for the Vowel, and for the Wisconsin Breast Cancer datasets.
6. Re-run the ANN models incorporating 10 hidden layers with 30 nodes. How much more time does it take to run a "bigger" neural network on the Wine dataset?
7. Build bagging models for the 2011 Gapminder, Wisconsin Breast Cancer, and Wine datasets.
8. Build random forest models for the 2011 Gapminder, Wisconsin Breast Cancer, and Iowa Housing datasets.
9. Build boosted models for the 2011 Gapminder, Wisconsin Breast Cancer, Wine, and Iowa Housing datasets.
10. Build classification models for the datasets
    - `GlobalCitiesPBI.csv`
    - `2016collisionsfinal.csv`
    - `polls_us_election_2016.csv`
    - `HR_2016_Census_simple.xlsx`
    - `UniversalBank.csv`.

    and/or any other datasets of interest. You may need to identify/define a categorical response variable first.

Chapter References

[1] 3Blue1Brown. *Deep Learning* .

[2] C.C. Aggarwal and C.K. Reddy, eds. *Data Clustering: Algorithms and Applications* . CRC Press, 2014.

[3] B. Boehmke and B. Greenwell. *Hands on Machine Learning with R* . CRC Press.

[4] M. Caudill. 'Neural Networks Primer, Part 1'. In: *AI Expert* 2.12 (Dec. 1987), pp. 46–52.

[5] N. V. Chawla et al. 'SMOTE: Synthetic Minority Over-sampling Technique '. In: *Journal of Artificial Intelligence Research* 16 (2002), pp. 321–357.

[6] F. Chollet. *Deep Learning with Python*. 1st. USA: Manning Publications Co., 2017.

[7] R.S. Sutton. 'Two Problems with Backpropagation and Other Steepest-Descent Learning Procedures for Networks'. In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum, 1986.

[8] Y. Dauphin et al. *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization*. 2014.

[9] N. Deng, Y. Tian, and C. Zhang. *Support Vector Machines: Optimization Based Theory, Algorithms, and Extensions*. CRC Press/Chapman and Hall, 2013.

[10] J. Friedman, T. Hastie, and R. Tibshirani. 'Additive Logistic Regression: a Statistical View of Boosting'. In: *Annals of Statistics* 28 (1998), p. 2000.

[11] D. Gershgorn. 'There's a glaring mistake in the way AI looks at the world '. In: (2017).

[12] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT press Cambridge, 2016.

[13] I.J. Goodfellow, J. Shlens, and C. Szegedy. 'Explaining and harnessing adversarial examples'. In: *ICLR* (2015).

[14] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction , 2nd ed.* Springer, 2008.

[15] T. Hofmann, B. Schölkopf, and A.J. Smola. 'Kernel Methods in Machine Learning'. In: *Annals of Statistics* 36.3 (2008), pp. 1171–1220.

[16] D. Hoftsadter. *Gödel, Escher, Bach: an Eternal Golden Braid*. Basic Books, 1979.

[17] R.V. Hogg and E.A Tanis. *Probability and Statistical Inference*. 7th. Pearson/Prentice Hall, 2006.

[18] G. James et al. *An Introduction to Statistical Learning: With Applications in R* . Springer, 2014.

[19] M.H. Kutner et al. *Applied Linear Statistical Models*. McGraw Hill Irwin, 2004.

[20] O. Leduc and P. Boily. 'Boosting with AdaBoost and Gradient Boosting '. In: *Data Action Lab Blog* (2019).

[21] R.M. Levenson et al. 'Pigeons (Columba livia) as Trainable Observers of Pathology and Radiology Breast Cancer Images'. In: *PLOS ONE* 10.11 (Nov. 2015), pp. 1–21. DOI: 10.1371/journal.pone.0141357.

[22] F. Provost and T. Fawcett. *Data Science for Business*. O'Reilly, 2015.

[23] D. Robinson. 'What's the difference between data science, machine learning, and artificial intelligence? '. In: *Variance Explained* (Jan. 2018).

[24] S. Ruder. *An overview of gradient descent optimization algorithms*. 2016.

[25] C. Sheppard. *Tree-Based Machine Learning Algorithms: Decision Trees, Random Forests, and Boosting*. CreateSpace Independent Publishing Platform, 2017.

[26] A. Turing. 'Computing Machinery and Intelligence'. In: *Mind* (1950).

[27] Wikipedia. 'Artificial Intelligence '. In: (2020).

[28] Wikipedia. 'Binary classification '. In: (2021).

[29] D.H. Wolpert. 'The Lack of A Priori Distinctions Between Learning Algorithms'. In: *Neural Computation* 8.7 (1996), pp. 1341–1390. DOI: 10.1162/neco.1996.8.7.1341.

[30] D.H. Wolpert and W.G. Macready. 'Coevolutionary free lunches'. In: *IEEE Transactions on Evolutionary Computation* 9.6 (2005), pp. 721–735. DOI: 10.1109/TEVC.2005.856205.

[31] D.H. Wolpert and W.G. Macready. 'No free lunch theorems for optimization'. In: *IEEE Transactions on Evolutionary Computation* (1997).

22 Focus on Clustering

by **Jen Schellinck** and **Patrick Boily**, with contributions from **Aditya Maheshwari**

In Chapter 19 (*Machine Learning 101*), we provided a (mostly) math-free general overview of machine learning.

Supervised learning methods can be presented in a formalism which generalizes statistical and regression analysis, and their performance are easy to evaluate; consequently, they have been studied extensively and often form the backbone of machine learning training.

On the other hand, apart from a select few classical models, **unsupervised learning** tasks are not usually presented with quite the same depth, primarily due to the vagueness which infect their core – some of the concepts are defined **ambiguously**; results validation is at times **elusive**, and the actionable applications of the outcomes are not always clear.

The interest in such methods and tasks (clustering and segmentation, association rules mining, link profiling, etc.) is mounting, however, with the recent advances in artificial intelligence and machine learning research. In this chapter, we describe various clustering algorithms, and discuss related issues and challenges.

This is a continuation of the treatment started in Chapter 20 (*Regression and Value Estimation*) and is a companion piece to Chapter 21 (*Focus on Classification and Supervised Learning*).

22.1 Overview

We introduced some of the basic notions of unsupervised learning in Chapter 19, *Introduction to Machine Learning*; in this chapter, we review some of these concepts in the context of clustering, discuss the problems of validation and model selection, and present some simple and sophisticated clustering algorithms.

22.1.1 Unsupervised Learning

In supervised learning (SL), we differentiate a dataset's **response variables** $Y_1, \ldots, Y_m$ from its **predictor variables** $X_1, \ldots, X_p$. Which variables are predictors and which are responses depend on the context – for some questions, a given variable could be a predictor, for others, a response.

Unsupervised learning tasks do away with the responses altogether, which means that **prediction** is off the table; the variables that would have

been deemed response variables in a SL framework are not necessarily removed from the dataset during the analysis – they are simply not viewed as an outcome to predict, and the predictor variables are just observation **features**.

In UL, the objective is to **identify** and **uncover interesting insights** about the dataset and the system that it represents (see Section 14.2.2, *Information Gathering*), such as:

- informative ways of visualizing the dataset (often associated with *Feature Selection and Dimension Reduction*, see Chapter 23);
- highlighting subgroups among the dataset's variables or observations (clustering), or
- finding links between variables (association rules mining, link profiling, etc.), say.

22.1.2 Clustering Framework

Clustering consists of a large family of algorithms and methods used to discover so-called **latent groups** in the datasets – natural groups that exist but have not been identified or labeled as such.

Clustering is a **subjective** analytical task; unlike classification and regression, clustering analysis does not have as "simple" a goal as predicting a response for a new observation based on historical data patterns, and there is no "solution key" against which to compare analysis results.

Applications:

- finding subgroups of breast and/or prostate cancer patients based on their gene expression measurements or their socio-demographic characteristics in order to better understand the disease and potential treatment side-effects;
- grouping products in an online shop based on ratings and reviews assigned by customers, or grouping customers based on their purchasing history, in order to make product recommendations;
- finding documents that apply to search queries, and finding similar queries to those entered by a user to increase the odds of finding the documents they are really looking for;
- identifying population segments to test various incentives for vaccination;
- etc.

In each of these cases, the **number** of these latent groups is unknown (and can in fact be taken as a true unknown of the problem). The **subjectivity** of unsupervised learning tasks may seem to be an insurmountable flaw: analysts attempting to find latent groups in a dataset, say, may obtain a different number of such groups, or assign different observations to their groups if their numbers are identical, without one of them being necessarily "wrong".[1]

In spite of this, clustering is a popular analytical task, in part because it is much easier (and cheaper), typically, to obtain **unlabeled data** than it is to obtain labeled data (against which supervised methods could be evaluated). A **cluster** is a subset of observations that all have something in common – they are **similar**, according to some measure of similarity.

1: Although it is conceivable that some of them could produce **sub-optimal** groups; see Section 22.3 for a detailed discussion on this topic.

Furthermore, a cluster's observations should be **dissimilar** to other clusters' observations.

Clusters do not necessarily need to be **disjoint** (as in so-called **hard** clustering) – in some cases, it might be sufficient to quantify the likelihood or the degree to which an observation belongs to a cluster (**soft** clustering).

The choice of a **similarity/dissimilarity measure** is also entirely **subjective**; there are contexts for which **proximity** could be used as a decent proxy for similarity, and others where it could not. Even in the former case, a **distance measure** (metric) has to be selected, and infinitely many choices are available to analysts.

Without **domain-specific considerations** (this requires thorough data and context understanding), the choice of measure is arbitrary; but understanding the data and the context does not guarantee that all reasonable analysts would agree on such a measure.

For instance, in any group of human beings, which of

> age, ethnic background, gender, postal code, sexual orientation, linguistic abilities, mathematical skills, career, social class, political affiliation, operating system preferences, educational achievements, hockey club fandom, etc.

is responsible for separating its members into "US" vs. "THEM" groups? Is it some combination of these characteristics? Are the groups fixed? Is everybody in the "US" group based on age also in the "US" group based on "gender"?

We could bypass the problem by creating more groups; given an age group and gender, we could create the clusters: "same age group and gender" (US), "same age group, different gender" (THEM_1), "different age group, same gender' (THEM_2)', "different age group and gender" (THEM_3).

It is clear how the process can be expanded to include more combinations of feature levels, but at the price of introducing an ever increasing number of clusters – how many "THEM" groups are too many for analysts or human brains to process?

Clustering algorithms are designed to try to model various aspects of this problem, but the latter's complexity gives rise to an enormous number of algorithms: at least 100 have been published, as of January 2022 [48]. Most of these belong to one of six main families [2]:

- **partitional** ($k-$means and variants, CLARA, etc.);
- **hierarchical** (AGNES, DIANA, BIRCH, etc.);
- **density-based** (DBSCAN, DENCLUE, OPTICS, etc.);
- **connectivity-based** (spectral and variants, etc.);
- **grid-based** (GRIDCLUS, STING and variants, etc.);
- **model-based** (mixture models, latent Dirichlet allocation, expectation-maximization, etc.).

As is the case for all analytical methods, some modifications are required when dealing with "**Big Datasets**", for **high-dimensional data**, or for specific **types of datasets**, such as stream data, network data, categorical

data, text and multimedia data, time series data, and so on. **Ensemble methods**, which combine various clustering results, can also prove useful.

Distance, Similarity, and Dissimilarity Measures Although the choice of how to interpret and compute **similarity** between observations is, to all intents and purposes, completely up to the analysts, all such measures must satisfy certain properties: they must take on

- large values for similar objects, and
- small (or even negative) values for dissimilar objects.

Dissimilarity measures function in the opposite manner. The **kernel functions**[2] of machine learning (see Section 21.4.2) are examples of similarity (or dissimilarity) measures, most notably the **Gaussian** (or radial) kernel

$$K_\gamma(\mathbf{x}, \mathbf{y}) = \exp(-\gamma \|\mathbf{x} - \mathbf{y}\|_2^2),$$

for a given $\gamma > 0$, for which points near one another (in the $\|\cdot\|_2$ sense) have a similarity measure $w = K(\mathbf{x}, \mathbf{y}) \approx 1$ (and thus are **similar**), and points far from one another have a similarity measure near 0 (and thus are dissimilar).

2: Formally, a **kernel** is a symmetric (semi-)positive definite operator $K : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}_0^+$. By analogy with positive definite square matrices, this means that $\sum_{i,j=1}^N c_i c_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0$ for all $\mathbf{x}_i \in \mathbb{R}^p$ and all $c_j \in \mathbb{R}_+$, and $K(\mathbf{x}, \mathbf{w}) = K(\mathbf{w}, \mathbf{x})$ for all $\mathbf{x}, \mathbf{w} \in \mathbb{R}^p$.

Some similarity measures are derived from **distance (metrics)** functions $d : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}_0^+$, with special properties:

1. $d(\mathbf{x}, \mathbf{y}) = 0 \iff \mathbf{x} = \mathbf{y}$;
2. $d(\mathbf{x}, \mathbf{y}) \geq 0$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^p$;
3. $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^p$;
4. $d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y})$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^p$.

In effect, distances are positive-definite symmetric functions $\mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}_0^+$ satisfying the **Triangle Inequality**. Commonly used distances include the:

- **Euclidean** distance $d_2(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2$;
- **Manhattan** distance $d_1(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_1$;
- **supremum** distance $d_\infty(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_\infty$;
- more general **Minkowski** distance $d_p(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_p$, for $p \geq 1$, of which the first three examples are special cases;
- and more esoteric distances such as the **Jaccard** distance for binary vectors, the **Hamming** distance for categorical vectors, the **Canberra** distance for ranked lists, the **cosine** distance for text data, **mixed** distances for mixed variables, and so on [13, 16].

Given a distance d, a common construction is to define the associated similarity measures

$$w = \ell - d, \quad w = \exp(-kd^2), \quad \text{or} \quad w = \frac{1}{1+d}.$$

Note that there are similarity measures that cannot be derived from distance measures, however.

Data Transformations Prior to Clustering Prior to clustering, it is crucial that the data be **scaled** (and potentially **centered**) so that none of the variables unduly influence the outcomes, or, as the expression prosaically puts it, so that we do not have to compare apples with oranges – if age in years and height in cm are dataset variables, a 10-unit difference in age is likely to be more significant (in real terms) than a 10-unit difference in height.

Putting everything on a (min, max) scale, for instance, guarantees that relative differences (relative to the distributions of each variables), and not absolute distances, play the important role. However, there are many ways to scale the data, and the scaling approach may have an effect on the clustering results.[3]

3: As we are sure you will not be surprised to find out, by this point – that is the way, with clustering: out of the frying pan and into the fire.

Common Difficulties There are issues related to clustering other than the vagueness we have already discussed:

- in many instances, the underlying assumption is that **nearness of observations** (in whatever metric) is linked with **object similarity**, and that **large distances** are linked with **dissimilarity**;
- the **lack of a clear-cut definition** of what a cluster actually is makes it difficult to validate clustering results;
- various clustering algorithms are **non-deterministic**;
- the number of clusters cannot usually be known before the analysis;
- even when a cluster scheme has been accepted as valid, a **cluster description** might be difficult to come by;
- most methods will find clusters in the data even if there are none;
- once clusters have been found, it is tempting to try to "explain" them, but that is a job for domain experts.

22.1.3 A Philosophical Approach to Clustering

In the context of artificial general intelligence,[4] clustering provides a basic way for **intelligences** to structure their experience of the world.

4: Think free-ranging robots, roughly speaking.

Clustering techniques can allow such machines to identify **object instances** in the world around them and then, on the basis of this identification, to identify or define **types of objects** by grouping together the object instances they have discovered. With this in mind, we can view creating **concepts** as the fundamental purpose of identifying groupings of similar datapoints; these concepts allow an intelligent agent (whether machine or person) to:

- work in **shorthand** when dealing with objects (i.e., it is easier to deal with 10 'cats' than with 10 unique objects), and
- make assumptions about the object instances in a cluster associated with the concept (if an object is a cat, then that object probably likes fish).

If the existence of some "**ground truth**" about what **should** be clustered together (and by extension what should be counted as a concept) can be presupposed, then regardless of what is currently known about that truth, neither the choice of algorithms nor of clustering algorithm parameters is wholly **subjective**, in the psychological sense of the term (where it has

the connotation of "coming from a person's experience", which tends to indicate that whatever it is that is being talked about does not exist separate from such an experience); choosing one algorithm over another (or one set of parameters over another) may lead to a "better" or "worse" reflection of the underlying ground truth.

But what counts as a ground truth? There are, of course, debates about this in philosophical circles. Suppose that **natural kinds** exist, that is to say, suppose that there is a **privileged** and **objectively essential** way in which objects are **properly grouped** in nature.

This assumption is very commonly made in the sciences, where uncovering or discovering **universal truths** about natural kinds of objects is a major objective. In such a case, natural kinds can count as a ground truth, with some clusterings more closely reflecting this reality than others.

The fact that the ground truth is not known by the clustering agent does not mean that it does not exist, or that it cannot be sought out using various techniques. Indeed, this is arguably what scientists do when they are using the **scientific method**; they do not know, *a priori*, which of their hypotheses are true or which are false, but they nonetheless engage in various techniques to try to get a better sense of what is true and false.

Even if the existence of natural kinds is rejected, it can still be the case that, relative to a particular circumstance, some clustering results are of **higher quality** than others. Or, stated in terms of goals, some clustering results could achieve a stated goal to a greater or lesser extent than others.

This does appear to be more subjective, in the sense that the goal, and the success of the outcome relative to the goal, are both defined by an individual or individuals, rather than being **independent** of them.

Outside of clustering, it is not unusual for people to create **contextual definitions** of what counts as 'good'. Consider as an example the concept of a 'good meal'. What qualifies as a good meal when camping in the backcountry is not the same as what counts as a good meal when staying at a four-star resort. Context matters.

This does not mean that it is impossible to have a bad meal under either of these circumstances, or that anything counts as a good meal – we do not use subjective in the stultifying post-modernist sense that there are no constraints whatsoever and everything is a social construct.[5]

5: This might be a bit of a straw-man definition of "post-modernist subjectivity", but perhaps not that much of one, in the final analysis; all things being equal, we lean more toward the objective side of things, both in nature and in data analysis.

Nonetheless, such situations are difficult to pin down or define in a rigorous fashion. Even if there were some more abstract or subjective sense in which something could be said to be **common to all types** of good meals – or to all types of good clustering results, in this analogy – it is difficult to imagine how this could be stated with any precision. This is, frustratingly, a typically human limitation to dealing with the world.

However, since the underlying objective of machine learning and artificial intelligence is to create machines with abilities similar to those of humans, perhaps it is worth trying to capture this less than rigorous approach within the context of machine learning.

Given this inherent lack of rigour, are there applied situations where clustering is **useful**? More concretely, suppose we desire to cluster furniture, based on data about the furniture. We could make **measurements**

of various kinds on physical objects, either selected randomly or haphazardly; perhaps, rather than working with the furniture itself, we could use a website catalogue in which each page showcases a particular type of furniture available for sale.

In this scenario, there may be one grouping (created by tagging and linking pages, for instance) of the furniture pages that allows users to **visit the website with maximum efficiency**, and another that helps the store **maximize its sales**.

Moreover, if we believe that natural kinds exist (which, as noted, is debated by philosophers but is a common assumption in science), there might also be one grouping of the furniture that best matches the underlying furniture natural kinds.

When considering outcomes relative to a particular situation, the most appropriate strategy for a particular clustering will depend, broadly speaking, on two considerations:

- the **chosen goal** it is intended to support, and
- the **underlying structure** of the data itself.

The first can be explicitly known and stated, but the second will likely not be known in advance, which leads to numerous **technical issues** when applying clustering algorithms.

A multitude of clustering algorithms can be applied to problems like the website furniture problems described above, and for each of those, many different parameter settings exist. Suppose six different clustering process are carried out in the case of the furniture website example and they generate six (potentially) different clustering outcomes.

Presumably, some will be more effective than others if the objective is to get people to spend a maximum amount of money on the online store. If the objective is to allow customers to make their purchases most quickly, the "optimal" clustering might not be the one that leads customers to spend the most money.

It is difficult to say ahead of time which of the six groupings will be the most effective one, in each of these cases. However, it might be possible to carry out **A/B testing** to determine which one is the most effective, once they have been generated. But can the A/B testing step be avoided?

If the applied goal (e.g. the goal to group furniture pages in order to maximize profits) can be operationalized more directly in terms of similarity and difference, then it can be more directly tied to clustering approaches.

If we think that the best way to increase sales is to have loose clusters where people are forced to browse a certain amount, while being exposed to somewhat similar (but still interesting) pieces of furniture to find what they want, it might be possible to select a clustering approach to generate clusters with this desirable property.

Returning for a moment to the less applied general artificial intelligence scenario introduced earlier, if the fundamental functionality of clustering is viewed as creating concepts, then it would seem to make sense to operationalize this goal in terms of creating groupings where the **observations** in a group are similar to each other and different from those

in other groups. In this context, a poor grouping would *de facto* be one where multiple observations are similar to those in other clusters, or very different from those in their own cluster. This could happen if a clustering process runs into technical difficulties, but it can also happen if there is no such **strongly grouped structure** in the data itself.

To eliminate the possibility that the problem is not linked with the chosen clustering procedure, one strategy is to use **multiple clustering techniques**, as well as **multiple parameter settings** for each technique.

If the issue remains, then we could conclude that it is likely that there is no good clustering structure in the data and by extension, in the objects being represented by the data.

Interested readers can get more information on clustering, as well as examples of applications, in [2, 1, 38, 9, 15, 32, 35, 33, 21, 37, 23, 5, 48, 17, 20, 40, 46, 31, 39, 19, 42, 29, 8].

22.2 Simple Clustering Algorithms

We start by briefly discussing two of the simplest clustering algorithms: $k-$**means** and **hierarchical clustering**.[6]

6: In this section, we borrow heavily from [20].

22.2.1 $k-$Means and Variants

One potential clustering objective could be to achieve minimal **within-cluster variation** – observations within a cluster should be very similar to one another, and the total variation over all clusters should be small.

Assume that there are k clusters in the (scaled) dataset

$$\mathbf{X}_{n\times p} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}.$$

Let $C_1, \ldots, C_k$ denote the set of indices in each cluster, so that

$$\{1, \ldots, n\} = C_1 \sqcup \cdots \sqcup C_k \quad \textbf{(hard clustering)};$$

we use the notation $\mathbf{x}_i \in C_\ell$ to indicate that observation i lies in cluster ℓ. The **within-cluster variation** $\mathrm{WCV}(C_\ell)$ measures the degree to which the observations in C_ℓ differ from one another.

The approach is partition-based; we look for a **partition** $\{C_\ell^*\}_{\ell=1}^k$ such that the total within cluster variation is minimized:

$$\{C_\ell^*\} = \arg\min_{\{C_\ell\}} \left\{ \sum_{\ell=1}^{k} \mathrm{WCV}(C_\ell) \right\}.$$

The first challenge is that there are numerous ways to define $\text{WCV}(C_\ell)$, and that they do not necessarily lead to the same results;[7] most definitions, however, fall in line with expressions looking like

$$\text{WCV}(C_\ell) = \frac{1}{(|C_\ell| - g)^\mu} \sum_{\mathbf{x}_i, \mathbf{x}_j \in C_\ell} \text{variation}(\mathbf{x}_i, \mathbf{x}_j),$$

where it is understood that $\text{variation}(\mathbf{x}, \mathbf{x}) = 0$.

Common choices for the **variation** include

$$\text{variation}(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 = \sum_{m=1}^{p} (x_{i,m} - x_{j,m})^2$$
$$\text{variation}(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_1 = \sum_{m=1}^{p} |x_{i,m} - x_{j,m}|;$$

these are typically used because of the ease of vectorizing the distance measurements, and not necessarily because they make the most sense in context.

With these choices, if all observations $\mathbf{x}$ within a cluster C are near one another, we would expect $\text{WCV}(C)$ to be small. The values of the parameter μ can be adjusted to influence the cluster sizes.

Traditionally, we use $\mu = 0$ (or $\mu = 1$) and $g = 0$, and the partition problem reduces to

$$\{C_\ell^*\} = \arg\min_{\{C_\ell\}} \left\{ \sum_{\ell=1}^{k} \frac{1}{|C_\ell|^\mu} \sum_{\mathbf{x}_i, \mathbf{x}_j \in C_\ell} \text{variation}(\mathbf{x}_i, \mathbf{x}_j) \right\}.$$

As an optimization problem, obtaining $\{C_\ell^*\}_{\ell=1}^k$ is NP−hard due to the **combinatorial explosion of possible partitions** $\{C_\ell\}_{\ell=1}^k$ when n is large.[8]

Algorithm: $k-$Means We can obtain a partition which is reasonably close to the optimal one,[9] without having to go through all possible partitions:

1. **randomly assign** a cluster number $\{1, \ldots, k\}$ to each observation in the dataset;
2. for each C_ℓ, compute the cluster **centroid**;
3. assign each observation to the cluster whose centroid is **nearest** to the observation;
4. repeat steps 2-3 until the clusters are **stable**.

Three choices need to be made in order for the algorithm to run:

- the **number of clusters** k in step 1;
- the **centroid computation measure** in step 2;
- the **distance metric** used in step 3.

The most common choice of centroid measure for numerical data is to use the vector of means along each feature of the observations in each cluster (hence, $k-$**means**); using other centrality measures yield different

7: As one would expect from clustering.

8: Computing the number of such partitions in general cannot be done by elementary means, but it is easy to show that the number is bounded above by n^k.

9: Hopefully...

methods (such as $k-$**medians**, for instance). For categorical data, the algorithm becomes $k-$**modes**.

The distance used in step 3 is usually aligned with the centroid measure of step 2 (and with the choice of a variation function in the problem statement): Euclidean for $k-$means, Manhattan for $k-$medians, Hamming for $k-$modes. Variants of these approaches may use a different random initialization step: the first iteration centroids may be selected randomly from the list of observations, say.[10]

10: Unfortunately, the clustering results depend very strongly on the initial randomization – a "poor" selection can yield arbitrarily "bad" (sub-optimal) results; $k-$means++ selects the initial centroids so as to maximize the chance that they will be well-spread in the dataset (which also speeds up the run-time).

Other variants indicate how to process computations in parallel (for Big Data, see Chapter 30) or for data streams (with an updating rule, see Chapter 28). The algorithm can be shown to converge to a **stable cluster assignment**, but there is no guarantee that this assignment is the **global minimizer** of the objective function; indeed, different initial conditions can find different **local minima**, i.e., different **clustering schemes.**

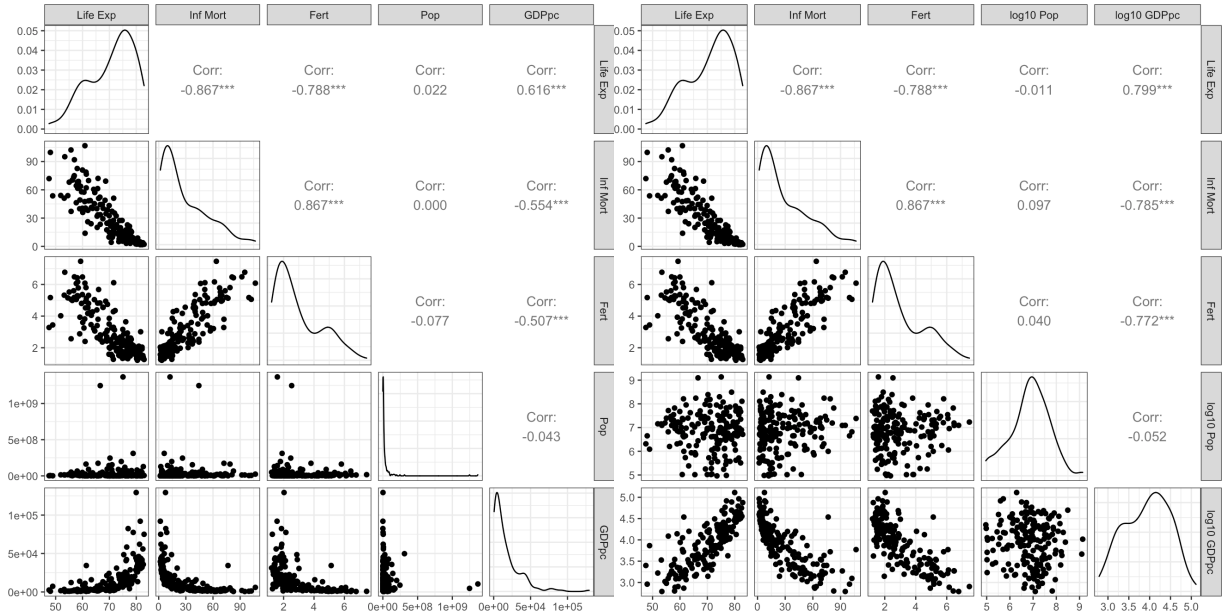
Example We have worked with the Gapminder data in Chapters 20 and 21; we will use a variant (`gapminder_all.csv`) to illustrate some of the notions in this chapter. The 2011 data contains observations on $n = 184$ countries, for the following variables:

- life expectancy (in years);
- infant mortality rate (per 1000 births);
- fertility rate (in children per woman);
- population (we use the logarithm for clustering), and
- GDP per capita (same).

```
library(dplyr)
library(tidyverse) # remove_rownames(), column_to_rownames()

gapminder.SoCL = read.csv("gapminder_all.csv",
    stringsAsFactors=TRUE)

gapminder.SoCL.2011  = gapminder.SoCL |>
  filter(time==2011) |>
  select(geo,
         population_total,
         income_per_person_gdppercapita_ppp_inflation_adjusted,
         life_expectancy_years,
         infant_mortality_rate_per_1000_births,
         children_per_woman_total_fertility) |>
  mutate(log10_pop=log10(population_total),
         log10_gdppc=log10(income_per_person_gdppercapita_ppp_inflation_adjusted)) |>
  na.omit() |>
  remove_rownames() |>
  column_to_rownames(var="geo")

colnames(gapminder.SoCL.2011)<- c("Pop","GDPpc","Life Exp",
         "Inf Mort", "Fert","log10 Pop","log10 GDPpc")
```

A scatter plot of the original and transformed datasets are shown below. We use the logarithm of the population and the logarithm of GDP per capita due to outlying observations in the population variable (China and India).

```
GGally::ggpairs(gapminder.SoCL.2011[,c(3:5,1:2)])
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)])
```



Throughout, we work with the **scaled** dataset.

```
gapminder.SoCL.2011.s <- data.frame(scale(gapminder.SoCL.2011[,c(3:7)]))
str(gapminder.SoCL.2011.s)
```

```
'data.frame':   184 obs. of  5 variables:
 $ Life.Exp   : num  -1.647 -1.151 0.635 0.377 1.366 ...
 $ Inf.Mort   : num  1.834 3.175 -0.602 -0.498 -0.959 ...
 $ Fert       : num  1.77 2.093 -0.404 -0.981 -0.7 ...
 $ log10.Pop  : num  0.742 0.635 0.921 -0.49 0.595 ...
 $ log10.GDPpc: num  -1.336 -0.304 0.672 -0.164 1.286 ...
```

The following function will allow us to plot the distributions of each of the variables in each of the clusters (plots appearing on the diagonal):

```
library(ggplot2)
my_dens <- function(data, mapping, ..., low = "#132B43",
                    high = "#56B1F7") {
  ggplot(data = data, mapping=mapping) +
    geom_density(..., alpha=0.7)
}
```

We run $k-$means for $k = 2, 3, 4$ and obtain the results shown in Figure 22.1.

```
set.seed(1) # for replicability
# k=2
gapminder.SoCL.2011.s.k2 = kmeans(gapminder.SoCL.2011.s,2,
       iter.max=250,nstart=1)
```

```
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
    aes(color=as.factor(gapminder.SoCL.2011.s.k2$cluster)),
    diag=list(continuous=my_dens))
table(gapminder.SoCL.2011.s.k2$cluster)
```

```
 1   2
64 120
```

```
# k=3
gapminder.SoCL.2011.s.k3 = kmeans(gapminder.SoCL.2011.s,3,
       iter.max=250,nstart=1)
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
    aes(color=as.factor(gapminder.SoCL.2011.s.k3$cluster)),
    diag=list(continuous=my_dens))
table(gapminder.SoCL.2011.s.k3$cluster)
```

```
 1  2  3
46 84 54
```

```
# k=4
gapminder.SoCL.2011.s.k4 = kmeans(gapminder.SoCL.2011.s,4,
       iter.max=250,nstart=1)
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
    aes(color=as.factor(gapminder.SoCL.2011.s.k4$cluster)),
    diag=list(continuous=my_dens))
table(gapminder.SoCL.2011.s.k4$cluster)
```

```
 1  2  3  4
26 50 61 47
```

The colours (cluster labels) are not used by the clustering algorithm – they are the **outputs**.[11] This next block shows the result of a different initialization for $k = 3$, leading to a different cluster assignment.

11: The actual cluster label value is entirely irrelevant.

```
set.seed(1234) # different initialization
gapminder.SoCL.2011.s.k3 = kmeans(gapminder.SoCL.2011.s,3,
       iter.max=250,nstart=1)
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
    aes(color=as.factor(gapminder.SoCL.2011.s.k3$cluster)),
    diag=list(continuous=my_dens))
table(gapminder.SoCL.2011.s.k3$cluster)
```

```
 1  2  3
56 74 54
```

Another $k-$means example is provided in Section 19.7.3.

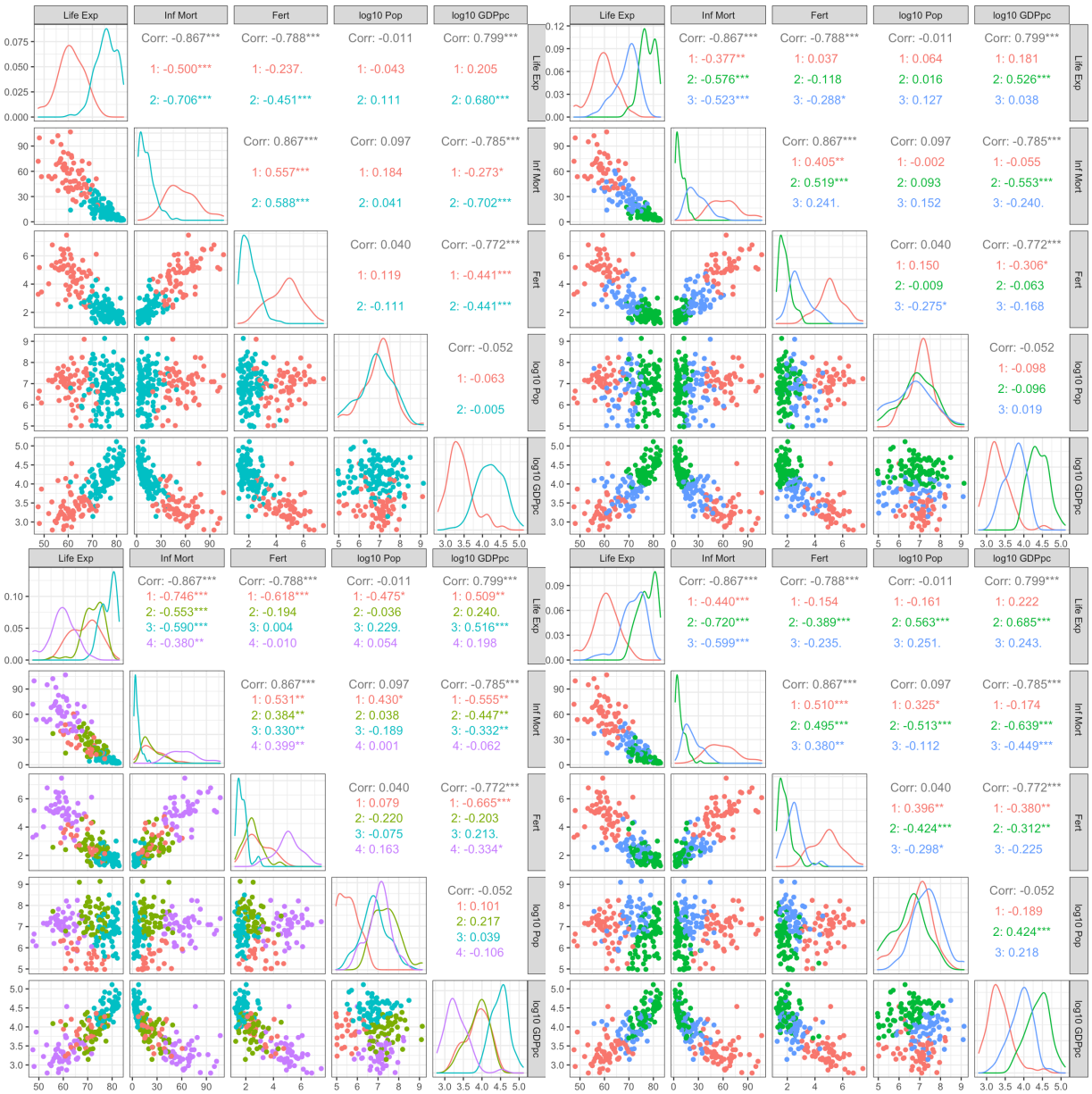


Figure 22.1: Realizations of $k-$means on the 2011 Gapminder data: $k = 2$ (top left); $k = 3$ (top right); $k = 4$ (bottom left); $k = 3$ with a different seed (bottom right). Are the two $k = 3$ clustering outcomes clearly distinct, to your eye?

22.2.2 Hierarchical Clustering

One of the issues surrounding the use of $k-$means (and its variants) is that nothing in the result of a single run of the algorithm indicates if the choice of k was a good one.[12]

12: The results might look good on a 2-dimensional representation of the data, but how do we know it could not look better?

Determining a "good" value of k can only be achieved by repeatedly running the algorithm over a range of "reasonable" values of k (to account for initialization variability), and by comparing the outputs using some of the methods discussed in Section 22.3. This process can be memory- (and time-)extensive.

Hierarchical clustering (HC) can sidestep this difficulty altogether by building a deterministic **global clustering structure** (for a given choice of parameters), from which we can select a specific number of clusters after the algorithm has converged; the advantage of this approach is that if we want to use a different number of clusters, we do not need to re-run the clustering algorithm – we simply read off the new clusters from the global clustering structure.

There are two main conceptual approaches:

- **bottom-up/agglomerative** (AGNES) – initially, each observation sits in its own separate cluster, which are then merged (in pairs) as the hierarchy is climbed, leading (after the last merge) to a large cluster containing all observations;
- **top-down/divisive** (DIANA) – initially, all observations lie in the same cluster, which is split (and re-split) in pairs as the hierarchy is traversed downward, leading (after the last split) to each observation sitting in its own separate cluster.

Both approaches are illustrated in Figure 22.2: the first one is an illustration of AGNES and DIANA. The corresponding hierarchical structure is shown in the second image; the dendrogram in the third.

In theory, the two approaches are equivalent;[13] in practice, we use AGNES over DIANA for anything but small datasets as the former approach runs in polynomial time,[14] whereas the latter runs in exponential time.

13: They produce the same hierarchical structure given a similarity metric and a **linkage strategy** (more on this later).

14: With respect to the number of observations.

With AGNES, the **clustering dendrogram** is built starting from the leaves, and combining clusters by pairs, up to the root, as in Figure 22.3.

Algorithm: AGNES The **global** clustering structure is built as follows:

1. each observation is assigned to **its own cluster** (there are n clusters, initially);
2. the two clusters that are the least dissimilar are merged into a **supra-cluster**;
3. repeat step 2 until **all of the observations** belong to a **single** large merged cluster (with n observations).

Three decisions need to be made in order for the algorithm to run:

- the choice of a **linkage strategy** in steps 2 and 3;
- the **dissimilarity measure** used in step 2;
- the **dissimilarity threshold** required to "cut" the dendrogram into clusters.

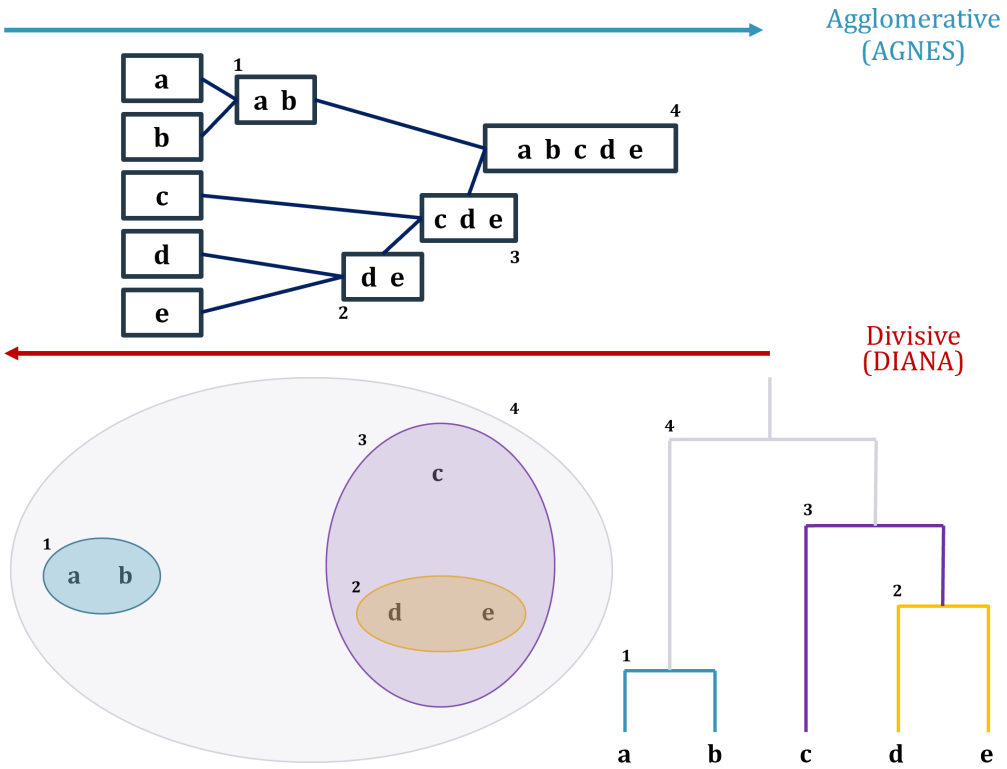


Figure 22.2: Conceptual representation of AGNES and DIANA on a simple artificial dataset.

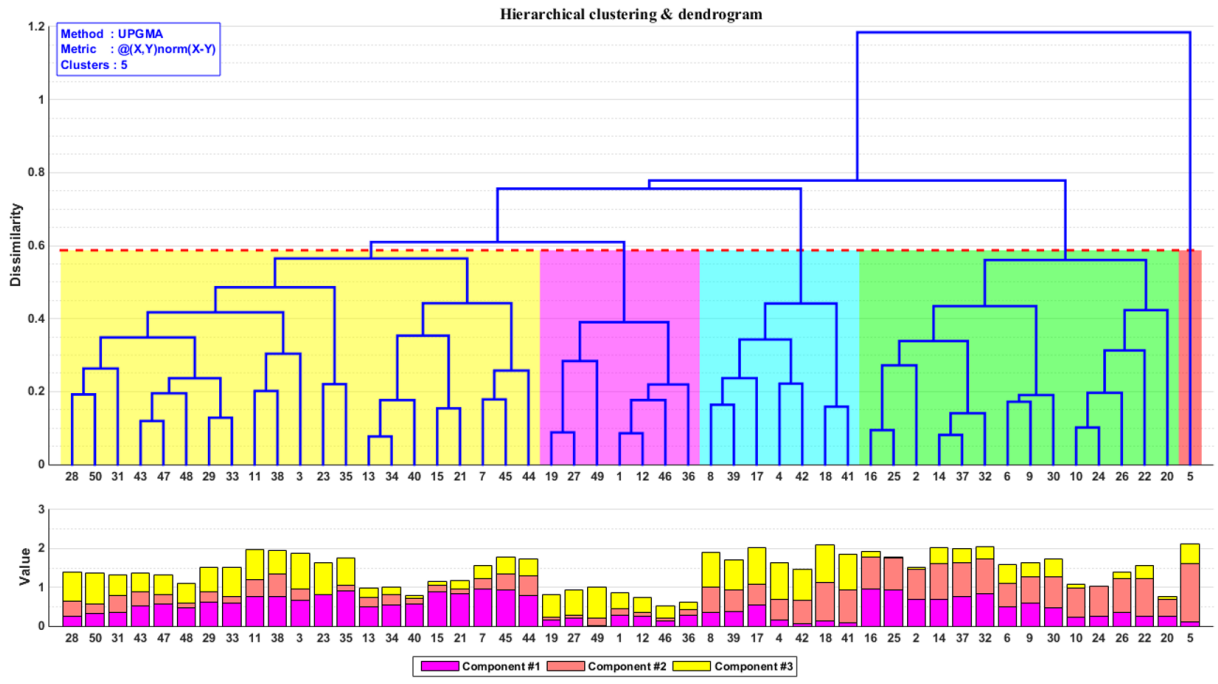


Figure 22.3: Cluster dendrogram for the hierarchical cluster structure of a dataset with 50 observations and 3 variables, with average linkage (UPGMA) and using the Euclidean distance as the dissimilarity measure [author unknown]. The dendrogram is cut at a dissimilarity level ≈ 0.6 so that 5 clusters emerge (ordered and coloured in red, magenta, blue, green, and red); the observations profiles are shown in the stacked bar chart and provide potential descriptions of the clusters (magenta = small total height, with mostly dominant 3rd components, say). Based on the profiles, we might also have elected to cut a slightly lower dissimilarity level (≈ 0.55), so that the yellow and green clusters would have been further split into two clusters apiece (between observations 35 and 13, and 30 and 10, perhaps?).

In Figure 22.3, the dataset is first split into $n = 50$ clusters; observations 13 and 34 are then found to be most similar, and merged into a single cluster, and the 50 observations are grouped into 49 clusters. The next two observations which are most similar are 14 and 37, which are themselves merged, so that there are 48 clusters at that level.

The process is continued until all observations are merged into a single cluster, leading to the global clustering structure (**clustering dendrogram**) for the dataset. In order to obtain actual clusters (as opposed to the global structure), we cut the dendrogram at the selected dissimilarity level, with the resulting groups of observations yielding the dataset clusters (5, in the example).

Increasing the dissimilarity threshold decreases the number of clusters, and *vice-versa*.

Linkage Strategy In the first AGNES stage, we compare all pairs of observations to determine which two are least dissimilar; these are then merged into a cluster.[15]

15: With n observations, there are $1 + \cdots + (n-1) = \frac{(n-1)n}{2}$ such pairs.

In the second stage, we must also compare each of the non-merged observations with a **cluster** of two observations to determine their dissimilarity (the other dissimilarities have been computed in the first stage and do not need to be computed anew).

In subsequent stages, we might also need to compare two clusters with any number of observations for overall similarity. How can this be achieved? Let A and B be clusters in the data, with $|A| = n_A$, $|B| = n_B$. The **dissimilarity** between A and B can be computed in multiple ways:

- **complete linkage** – the **largest** dissimilarity among all pairwise dissimilarities between the observations in A and those in B ($n_A n_B$ computations);
- **single linkage** – the **smallest** dissimilarity among all pairwise dissimilarities as in complete linkage;
- **average linkage** – the **average** dissimilarity among all pairwise dissimilarities as in complete (or single) linkage;
- **centroid linkage** – the dissimilarity between the **centroids** of A and B (found using whatever method is appropriate for the context);
- **Ward's method** (and its variants) uses any reasonable **objective function** which reflects domain knowledge [4, 47];
- etc.

The choice of a **linkage strategy** (and of a dissimilarity measure) affects not only how clusters are compared and merged, but also the **topology** (shape/structure) of the resulting dendrogram (are the clusters tight, loose, blob-like, etc.). The various linkage strategies are illustrated in Figure 22.4, assuming Euclidean dissimilarity.

Example We show the results of hierarchical clustering on the Gapminder 2011 data, using complete linkage and Euclidean dissimilarity, and Ward D linkage and the maximum dissimilarity. In each case, we consider $k = 2, 3, 4$ clusters.[16] The cluster charts are in Figure 22.5.

16: **Reminder:** we work on the scaled data. Assume that `library(ggplot2)` has already been loaded.

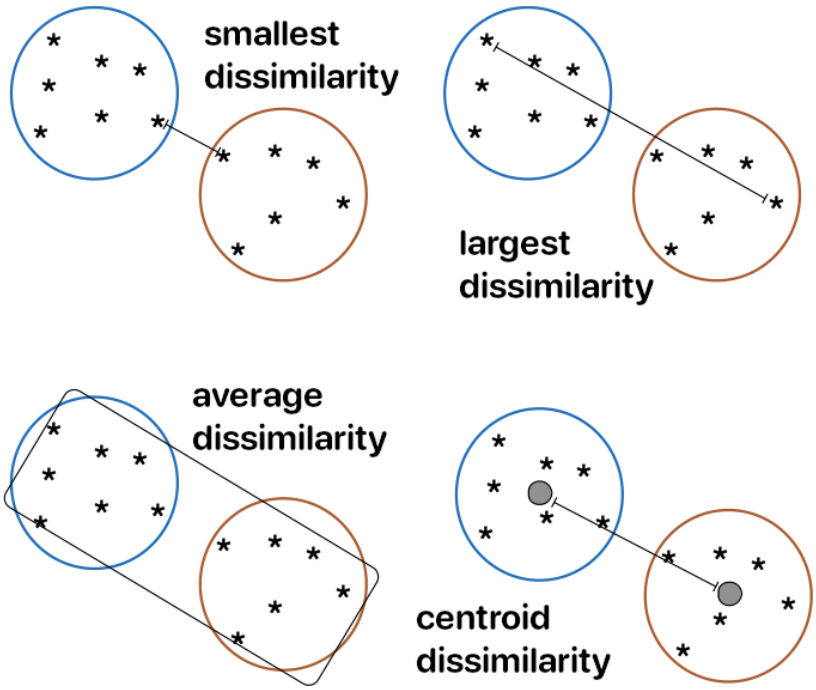


Figure 22.4: Conceptual notions of linkage, assuming Euclidean dissimilarity.

We first need to create the AGNES structure for the data using complete linkage and Euclidean dissimilarity.

```
# global, complete, Euclidean
par(cex=0.45)
hclust.gapminder.SoCL.2011 <- hclust(dist(gapminder.SoCL.2011.s))
plot(hclust.gapminder.SoCL.2011, hang = -1, cex=0.7,
     main = "Gapminder 2011 Data \n HC - Global Structure
     - Euclidean - Complete", ylab="")
```

Let us find the breakdown for $k = 2, 3, 4$ clusters for complete linkage and Euclidean dissimilarity.

```
# k=2, complete, Euclidean
par(cex=0.45)
hclust.gapminder.SoCL.2011 |> as.dendrogram() |>
  dendextend::set("branches_k_color", value = c("red", "blue"), k = 2)  |>
  plot(main = "Gapminder 2011 Data \n HC - 2 clusters - Euclidean - Complete")

hclust.gapminder.SoCL.2011 |> as.dendrogram() |>
    dendextend::rect.dendrogram(k=2, border = 8, lty = 5, lwd = 2, lower_rect=0)

GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
    aes(color=as.factor(cutree(hclust.gapminder.SoCL.2011, k = 2))),
    diag=list(continuous=my_dens))

table(cutree(hclust.gapminder.SoCL.2011, k = 2))
```

```
1   2
66 118
```

```
# k=3, complete, Euclidean
par(cex=0.45)
hclust.gapminder.SoCL.2011 |> as.dendrogram() |>
  dendextend::set("branches_k_color",
      value = c("red", "blue", "darkgreen"), k = 3)  |>
  plot(main = "Gapminder 2011 Data \n HC - 3 clusters -
      Euclidean - Complete")

hclust.gapminder.SoCL.2011 |> as.dendrogram() |>
    dendextend::rect.dendrogram(k=3, border = 8, lty = 5,
        lwd = 2, lower_rect=0)

GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
    aes(color=as.factor(cutree(hclust.gapminder.SoCL.2011,
        k = 3))), diag=list(continuous=my_dens))

table(cutree(hclust.gapminder.SoCL.2011, k = 3))
```

```
 1  2  3
66 24 94
```

```
# k=4, complete, Euclidean
par(cex=0.45)
hclust.gapminder.SoCL.2011 |> as.dendrogram() |>
  dendextend::set("branches_k_color",
      value = c("red", "blue", "darkgreen", "gray"), k = 4)  |>
  plot(main = "Gapminder 2011 Data \n HC - 4 clusters -
      Euclidean - Complete")

hclust.gapminder.SoCL.2011 |> as.dendrogram() |>
    dendextend::rect.dendrogram(k=3, border = 8, lty = 5,
        lwd = 2, lower_rect=0)

GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
    aes(color=as.factor(cutree(hclust.gapminder.SoCL.2011,
        k = 4))), diag=list(continuous=my_dens))

table(cutree(hclust.gapminder.SoCL.2011, k = 4))
```

```
 1  2  3  4
35 24 94 31
```

Notice how the number of observations in each cluster follows a hierarchical structure: when we go from $k = 2$ to $k = 3$ clusters, the new cluster is a subset of one of the old clusters (and similarly when we go from $k = 3$ to $k = 4$).

We can see how the results change when we use a different distance metric (maximum) and a different linkage strategy (Ward D): the line `hclust.gapminder.SoCL.2011 <- hclust(dist(gapminder.SoCL.2011.s))`is substituted throughout by `hclust.gapminder.SoCL.2011.2 <- hclust(dist(gapminder.SoCL.2011.s,method="maximum"), method="ward.D")`.

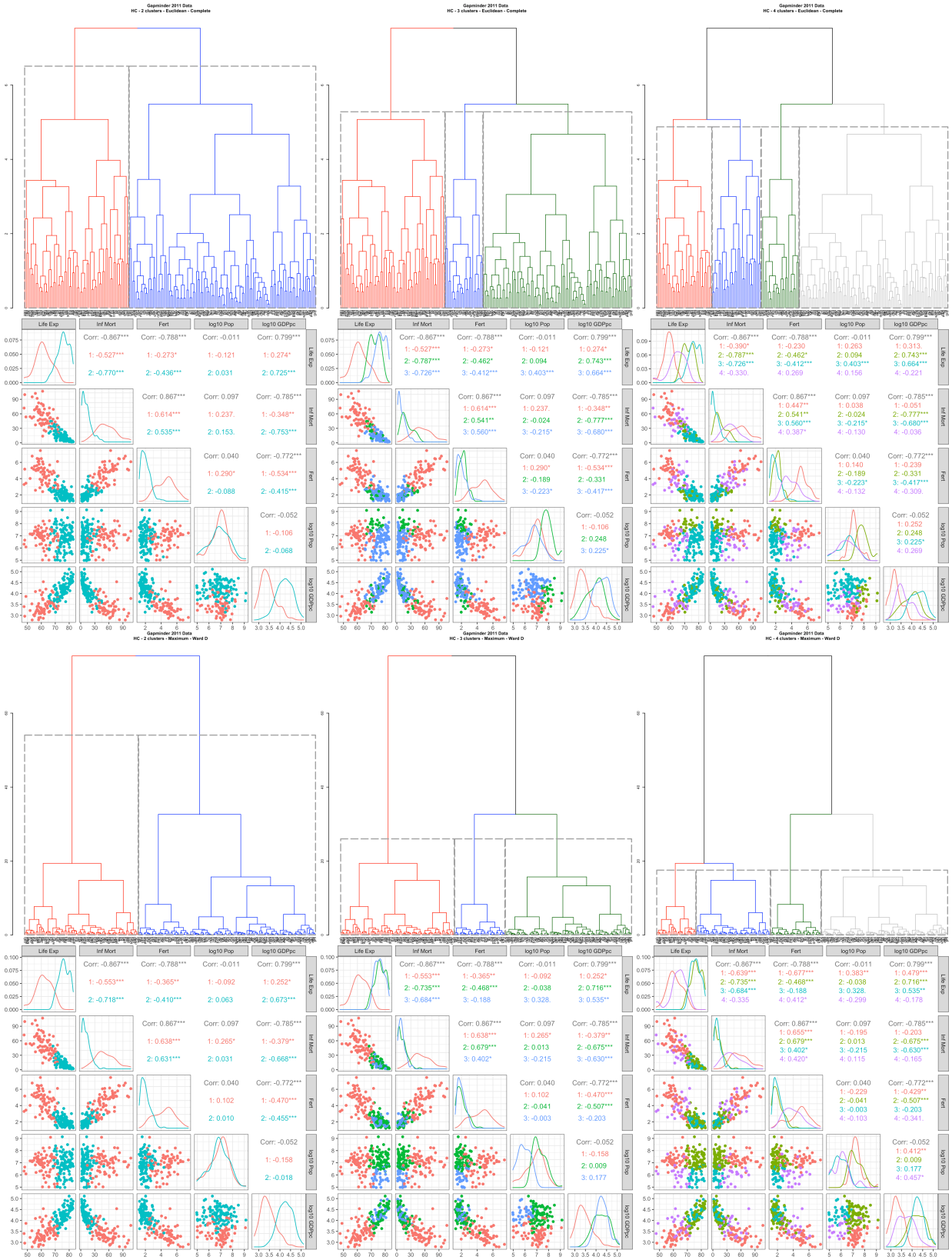


Figure 22.5: Realizations of hierarchical clustering (AGNES) on the 2011 Gapminder data: complete linkage, Euclidean dissimilarity for $k = 2, 3, 4$ clusters (top 2 rows); Ward D linkage, maximum dissimilarity for $k = 2, 3, 4$ clusters (bottom 2 rows).

22.3 Clustering Evaluation

Hierarchical clustering (HC) and $k-$means (and its variants) both attempt to separate the data into **natural groups**, using different conceptual approaches; $k-$means tries to minimize within-cluster variation while HC builds a global clustering structure.

We have discussed some of their shortcomings in the previous section; the fact that they may yield different clustering outcomes depending on the choices made along the way (initialization, similarity/dissimilarity measures, linkage strategy, number of clusters, etc.) reinforces the notion that unsupervised learning is **difficult**.

We will discuss advanced algorithms that sidestep some of these issues in Section 22.4, but we make an important observation in the meantime: a hallmark of clustering is that whenever a new approach manages to overcome a previously-identified difficulty, it usually does so at the cost of introducing holes in hitherto sound aspects.

We may thus not be able to ever find a "best" clustering approach/outcome,[17] but is it possible to identify which of several clustering scheme is "**optimal**" (or best-suited for an eventual task)?

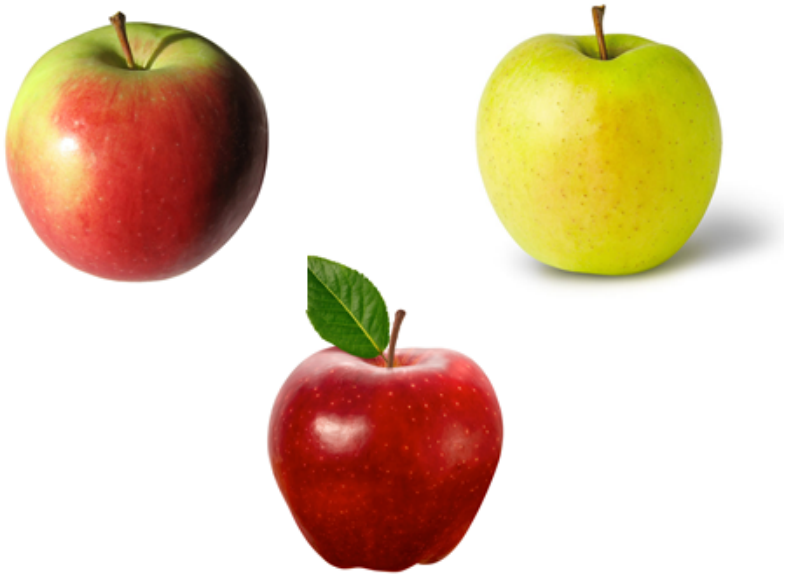
17: An updated take on the No Free Lunch theorem, perhaps? [49]

22.3.1 Clustering Assessment

In machine learning, clustering is defined as grouping objects based on their over-all similarity (or dissimilarity) to each other.[18] It can be tempting to focus on just one or two attributes (especially for visual or "eyeball" clustering), but keep in mind that even if we were to focus on one or two particular attribute, all of the other attributes must still come along for the ride.

18: Note that each object has multiple dimensions, or attributes available for comparison.

For instance, consider the objects shown below.



What is the same about these objects? What is different? Do they belong in the same group? If not, how many groups are there?

As a start, they are all pictorial representations of apples;[19] they all possess stems, and appear to be of similar size. On the other hand, two of them are (mostly) red while the other is green(ish); one of the stems has a leaf while the other two do not, and one of them is spherical, while the other two are "tapered" at the bottom.

19: While we cannot forget that they are not actual apples, we will assume that this is understood and simply refer to the objects as fruit, or apples.

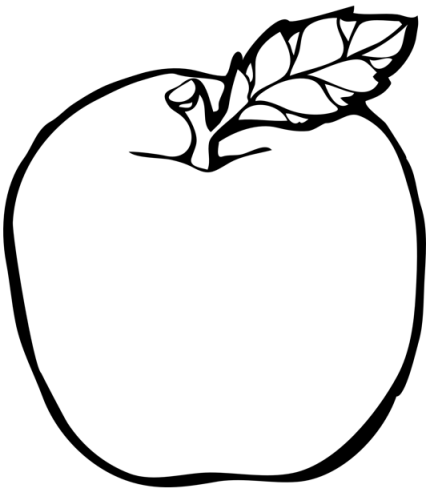
While we do recognize them all as apples, we could make the argument that each of them is unlike the other two (and thus also that each of them is similar to exactly one of the other two).

Fruit Image Dataset In order to appreciate the challenges presented by clustering validation, it will be helpful to relate the concepts to something tangible. We will explore some of these notions through an artificial dataset of 20 fruit images (see Figure 22.6):

- are there right or wrong clusterings of this dataset?
- are there multiple possible 'natural' clusterings?
- could different clusterings be used for different tasks?
- will some clusterings be of (objectively) higher quality than others?

Key Notions At a fundamental level, clustering relies on the notion of **representativeness**; ideally, the essence of **instances** (observations) in a cluster would be faithfully captured by the cluster **concept** (examplar, representative), and differentiated from those of other clusters by the same token.

As an example, the image below is a concept for "apples":



as is the Mirriam-Webster definition:

> "The fleshy, usually rounded red, yellow, or green edible pome fruit of a usually cultivated tree (genus Malus) of the rose family."

Of course, this is not *all* that an apple is, but most people who have seen or eaten an apple at some point in the past will have no trouble recognizing what is being alluded to by the concept, even if the corresponding mental image differs from one person to the next.

The cluster concepts offer a **generalized representation** – they capture "something" of their corresponding cluster's instances. For a given cluster, then, can we clearly identify a concept capturing its (and solely its) essence? If so, does that make the entire clustering scheme a good one?

For machine learning purposes, we use **signature vectors** to represent **instance properties**. Each vector element represents an instance **attribute**; the element's value is the **measured value** of the corresponding object's property (for instance, the colour of the apple).

The apple below, as an example, could perhaps be described by the signature vector

$$(12, 9.12, \text{tapered}, \text{golden delicious}),$$

where the components are the instance's **colour** (ordinal), **height** (continuous), **shape**, and **variety** (both categorical).[20]

20: An important consideration, from a general data science perspective, is whether the signature vector provides a **sufficient description** of the associated object or whether it is too crude to be of use. This is usually difficult to ascertain prior to obtaining analysis results, and comparing them to the "reality" of the underling system (see Chapters 13 and 14 for details).

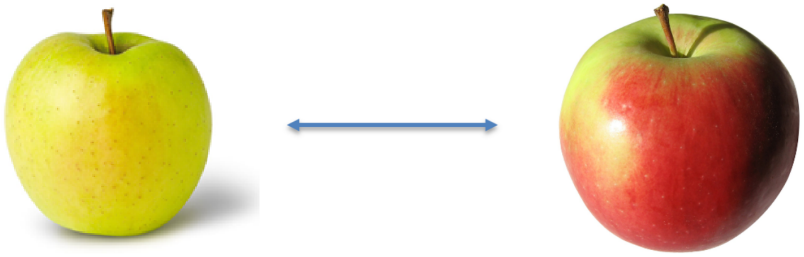
Signature vectors are used to compare objects (**instance-to-instance relationships**); such comparisons could yield, among others, a measure of **similarity** between instances.

While similarity can be measured against a single **dimension** (component), the comparisons of interest for clustering task require an overall similarity measure, **across all dimensions**. We would compare the two apples below, say, by comparing their signature vectors

$$\mathbf{v}_1 = (12, 9.12, \text{tapered}, \text{golden delicious})$$
$$\mathbf{v}_2 = (2, 10.43, \text{spherical}, \text{macintosh})$$

with the help of some similarity measure $w(\mathbf{v}_1, \mathbf{v}_2)$.[21]

21: Keep in mind that different similarity measures may yield various results, in some cases showing the two apples to be similar, in others to be dissimilar.

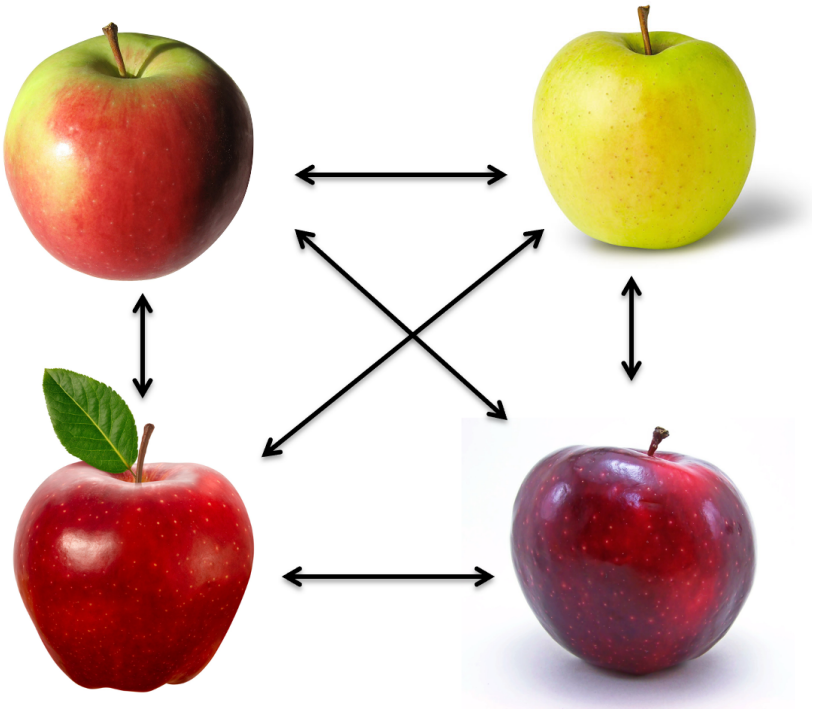


As we have discussed in Section 22.1, the use of a **distance measure** (or metric) is a popular strategy for defining how similar (or dissimilar) two objects are to each other. Importantly, a distance takes into account all of the properties of the objects in question, not just a few of them. Traditionally, only numeric attributes are allowed as input (see Chapter 26 for an in-depth discussion of distance metrics), but it is technically possible to convert categorical attributes to numeric ones, or to define **mixed distances**.[22]

22: While the moniker "distance" harkens back to the notion of Eulidean (physical) distance between points in space, it is important to remember that the measurements refer to the distance between the associated signature vectors, which do not necessarily correspond to their respective physical locations.

In the **clustering framework**, we are often interested in all pairwise similarities between objects, not just in the similarity between two objects, which is to say that pairs of objects are not solely interesting in and of themselves, but also **in relation to other pairs of objects**.

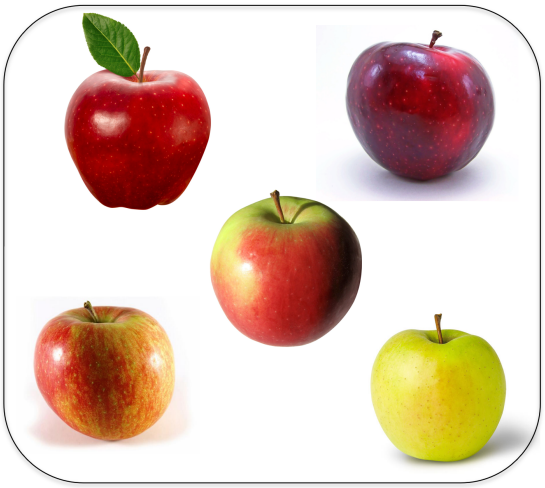
In a dataset with 4 objects, for instance, we might require the computation of (up to) 6 pairwise similarities (as shown below).



As is the case with objects, **clusters** also have **properties**. These could include:

- the number of instances in a cluster;
- similarity statistics across instances within a cluster (minimum, maximum, average, median, standard deviation, mean absolute deviation, variance, etc.);
- the cluster representative, which may be an actual instance, or an amalgamation of multiple instances (**exemplar**).

How many instances are there in the cluster at the top of the next page, for instance? What pair of observations is most similar? The least similar? What are the similarity values? Which instance is most representative?



We can also define **cluster-to-instance** relationships. A specific instance can be:

- compared to a cluster representative, and/or
- compared to specific instances in a cluster (as in instance-to-instance relationships), such as the most similar instance or the most distant instance.

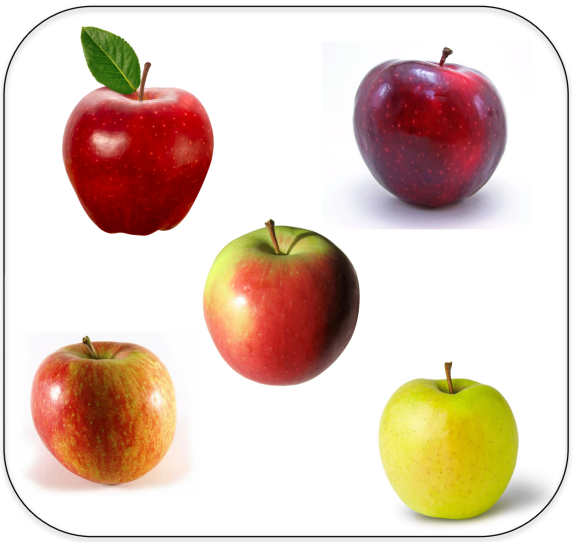
This allows for **membership** questions: is the green apple similar to the cluster below? Does it belong in the cluster, or is it most likely to belong to another cluster? Or perhaps to no cluster in particular?

Finally, we might be interested in **cluster-to-cluster** relationships, where we compare cluster-level properties, such as:

- number of instances;
- **within-cluster** similarities;
- cluster representatives.

To these, we can also add **between-cluster** (or across-cluster) similarities, as a way to determine if the instances are notably different from one cluster to the next. This allows for **validity** questions: are the two clusters

below significantly different? Should they be joined into a mega-cluster? Does it make sense to have them as separate clusters in the dataset?



How would we qualify the clustering outcome of Figure 22.7, for instance, as it relates to colour, height, and shape? Could there be clusterings of higher quality? Of lower quality? How could this be quantified?

Cluster and instance comparisons can be combined in many different ways, which can then be used to generate a vast number of **clustering validation functions**.

The central cluster validation question becomes: what can these tell us about the quality of a particular clustering outcome relative to some objective criteria about "good" clustering schemes (**internal validation**), to another clustering option (**relative validation**), or to external information (**external validation**)?

Clustering Quality Measures In general, clustering involves two main activities:

- **creating/building** the clusters, and
- **assessing their quality**, individually and as a whole.

From a practical perspective, clustering requires two functions: one which assigns each instance to a cluster,[23] and one which assigns each clustering scheme to a **cluster quality measurement**.[24]

An illustration is provided in Figure 22.8, on an artificial dataset containing two variables, with dissimilarity between instances given by the corresponding Euclidean distance.

23: Or in the case of soft clustering, assign each instance a "probability" of belonging to each cluster.

24: The similarity matrix is typically required at both stages.

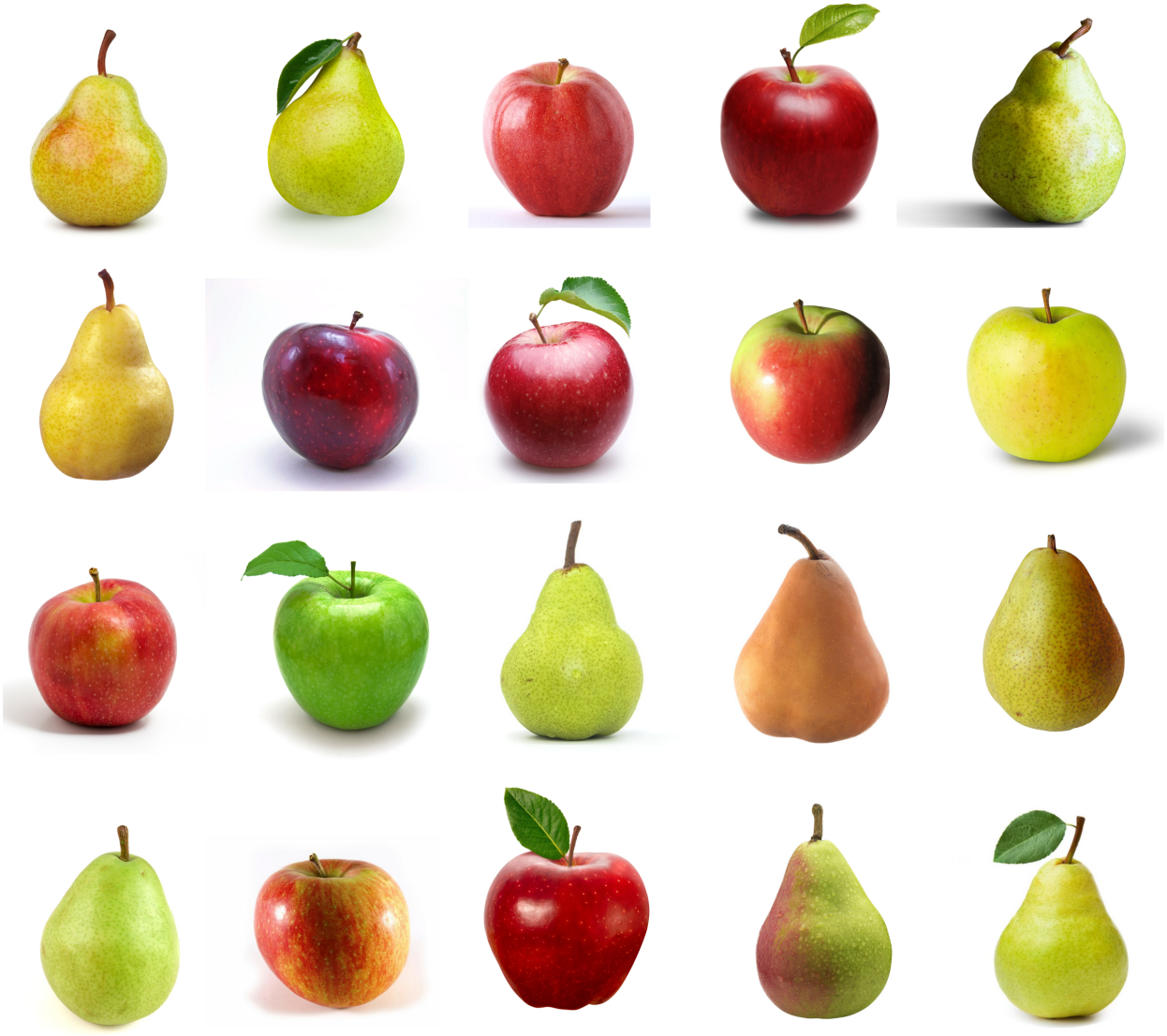


Figure 22.6: Toy dataset with which the key concepts of clustering validation will be illustrated.

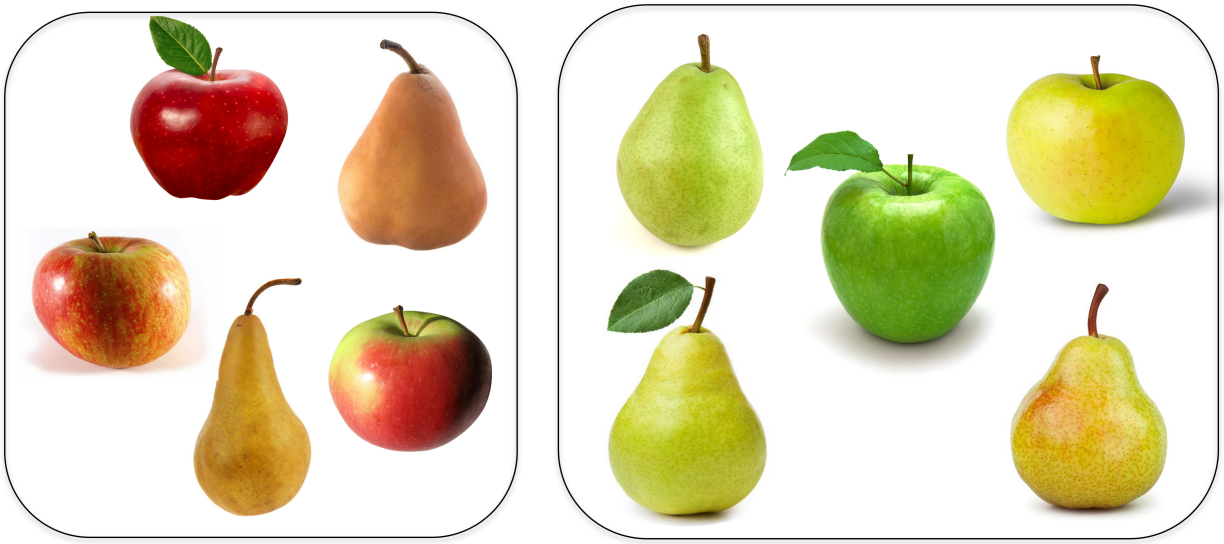


Figure 22.7: Two clusters in a subset of the fruit image toy dataset.

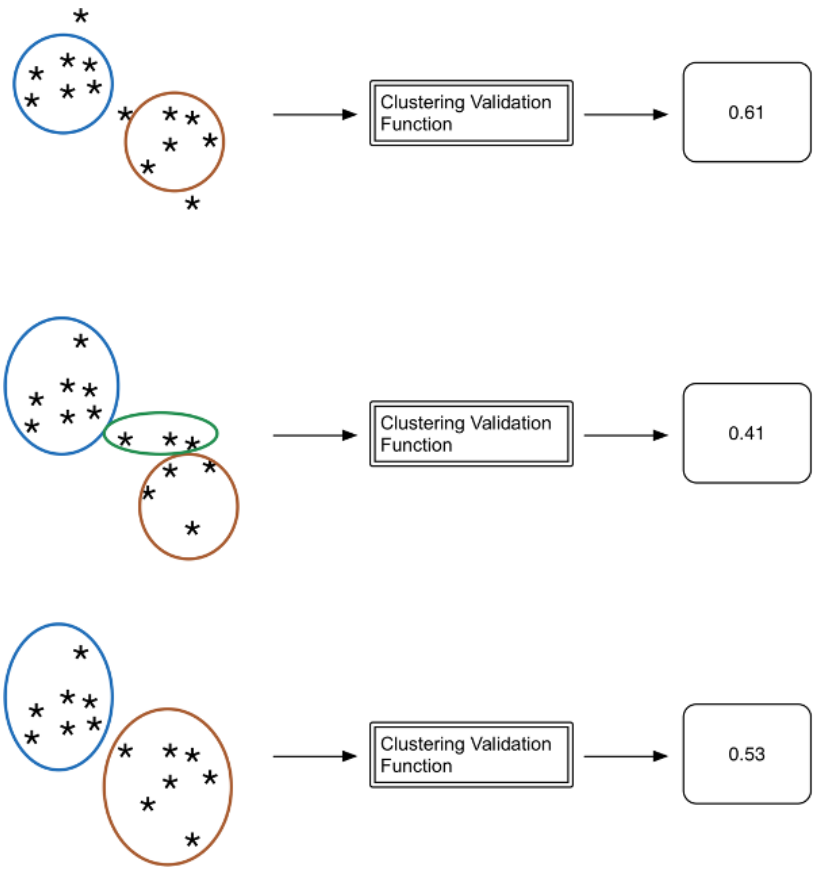


Figure 22.8: Cluster quality measurements on an artificial dataset.

We obtain three different clustering schemes, and their quality is assessed with the help of some clustering validation function:[25]

- **top** – two clusters are found in the data (with outliers), and the quality of the clustering is assessed as 0.61;
- **middle** – three clusters are found (no outliers), with quality assessment at 0.41;
- **bottom** – two clusters are found (no outliers), with quality assessment at 0.53.

With this choice of clustering validation function, the top scheme would be preferred, followed by the bottom scheme; the middle one brings up the rear. We have already mentioned the abundance of clustering algorithms [48]; it will come as no surprise that a tremendous number of clustering validation function in practice as well (for much the same reasons as those discussed in Section 22.1.2).

They are, however, all built out of the basic measures relating to instance or cluster properties we have already reviewed, as illustrated schematically in Figure 22.9:

- **instance properties**;
- **cluster properties**;
- **instance-to-instance relationship properties**;
- **cluster-to-instance relationship properties**, and
- **cluster-to-cluster relationship properties**.

25: The specifics of that function are not germane to the current discussion and so are omitted.

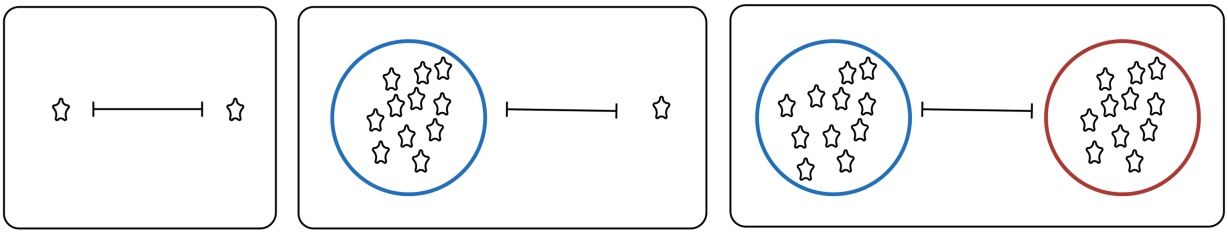
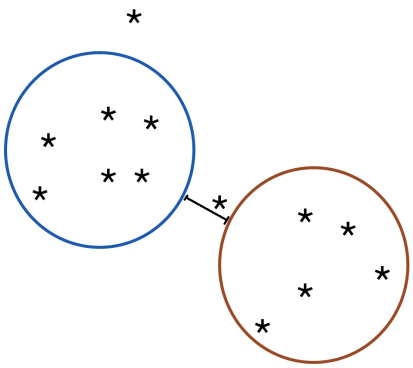


Figure 22.9: Schematic illustrations of various instance/cluster properties and relationships.

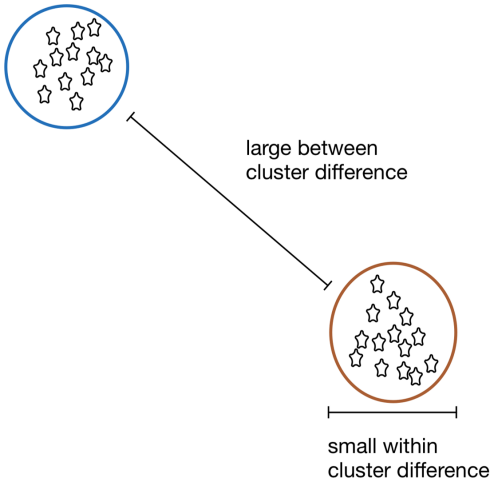
Internal Validation Context is quite relevant to the **quality** of a given clustering result. But what if no context is readily available? **Internal validation** methods attempt to measure cluster quality objectively, without context.[26]

26: "Clustering validation" suggests that there is an ideal clustering result against which to compare the various algorithmic outcomes, and all that is needed is for analysts to determine how much the outcomes depart from the ideal result. "Cluster quality" is a better way to refer to the process.

We could elect to validate the clustering outcome using only the properties of the obtained clusters, such as, say, the distance between all clusters: if the average between-cluster distance is large, we might feel that there is some evidence to suggest that the resulting clusters provide a good segmentation of the data into **natural groups**, as in the image below.



Alternatively, we might validate cluster quality by tempering the average between-cluster distance against the average within-cluster distance between the instances, which would reward "tight" and "isolated clusters", as opposed to simply "isolated" ones, as below.



There are multiple ways of including both between-cluster and within-cluster similarities in a **cluster quality metric** (CQM): both the **Davies-Bouldin index** and **Dunn's index** do so, to name but two examples. The broad objectives of clustering remain universal: instances within a cluster should be similar; instances in different clusters should be dissimilar.

The problem is that there are many ways for clusters to deviate from this ideal: in specific clustering cases, how do we weigh the "good" aspects (such as high within-cluster similarity) relative to the "bad" ones (such as low between-cluster separation)?

Other internal properties and relationships can also be used and combined in various ways, leading to an embarrassment of riches when it comes to CQMs. The following indices are all available in the `R` package `clusterCrit`, for instance [10]:

- Ball-Hall
- Banfeld-Raftery
- C
- Calinski-Harabasz
- ***Davies-Bouldin**
- Det Ratio and LogDetRatio
- ***Dunn**
- Baker-Hubert Gamma
- GDI
- Gplus
- KsqDetW
- McClain-Rao
- PBM
- Point-Biserial
- Ratkowsky-Lance
- Ray-Turi
- Scott-Symons
- SD
- SDbw
- ***Silhouette**
- Tau
- TraceW and TraceWiB
- Wemmert-Gancarski
- ***WSS**
- LogSSRatio
- Xie-Beni

While it is useful to have access to so many CQMs, we should remain aware that the No-Free Lunch theorem is still in play: none of them is universally superior to any of the others.[27]

27: Given that all of them are supposedly provide context-free assessments of clustering quality, that is problematic (although emblematic of unsupervised endeavours).

In practice, certain approaches (**weightings**) may prove more relevant given the eventual specific analysis goals, but what these could prove to be is not always evident from the context; consequently, we recommend assessing the quality of the clustering outcomes using a variety of CQMs. Studies have been performed to compare a large number of internal validation measures, relative to one another and against human evaluation; generally speaking, variants of the **silhouette index** performed reasonably well (but see previous NFLT comment) [44, 26].

One possible interpretation of these results is that internal validation *via* CQMs may be providing information about clustering across all contexts, and that it is usually easier to identify clustering outcomes that clearly miss the mark than it is to objectively differentiate amongst "good" outcomes, for reasons that are not fully understood yet.

Details on the CQMs are readily available from a multitude of sources (see [2, 26, 44]); we provide more information for 4 CQMs:

- (within) sum of squared error;
- Davies-Bouldin;
- Dunn, and
- silhouette;

Within Sum of Squares Let $\mathscr{C} = \{C_1, \ldots, C_K\}$ be the K clusters obtained from a dataset $\mathbf{X}$ *via* some algorithm $\mathscr{A}$. Denote the **centroid** (or some other central representative) of cluster C_k by $\mathbf{c}_k$. The **(within) sum of error** for $\mathscr{C}$ is

$$\text{WSE} = \sum_{k=1}^{K} \sum_{\mathbf{x} \in C_k} \text{dissimilarity}(\mathbf{x}, \mathbf{c}_k).$$

The dissimilarity is often selected to be the **square of the Euclidean distance** (thence "sum of squared error"), but the choice of the Euclidean distance (and of the square exponent) is arbitrary – it would make more sense, in practice, to use a dissimilarity related to the similarity measure used by $\mathscr{A}$.

Note that WSE decreases with the number of clusters K, and optimality is obtained at **points of diminishing returns** (see the next section for details); from a validation perspective, it might be easier to interpret the **(within) average error**:

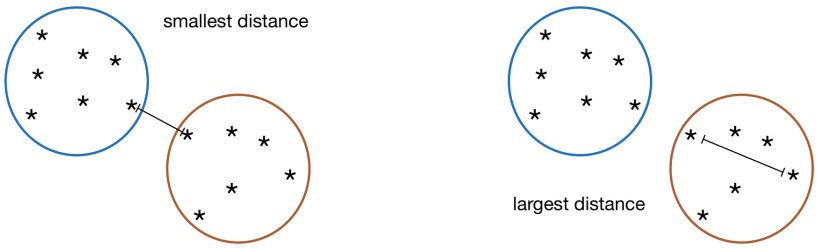
$$\text{WAE} = \text{Avg}_{k=1}^{K} \left\{ \text{Avg}_{\mathbf{x} \in C_k} \left\{ \text{dissimilarity}(\mathbf{x}, \mathbf{c}_k) \right\} \right\} = \text{Avg}_{k=1}^{K} \left\{ s_k \right\}.$$

Davies-Bouldin Index With s_k, $k = 1, \ldots, K$ as above, the **Davies-Bouldin index** (DBI) is defined as

$$\text{DBI} = \frac{1}{K} \sum_{k=1}^{K} \max_{\ell \neq k} \left\{ \frac{s_k + s_\ell}{\text{dissimilarity}(\mathbf{c}_k, \mathbf{c}_\ell)} \right\}.$$

If the clusters $\{C_k\}$ are **tight** and **dissimilar to one another**, we expect the numerators $s_k + s_\ell$ to be small and the denominators dissimilarity$(\mathbf{c}_k, \mathbf{c}_\ell)$ to be large, so that the DBI would be **small**.

Dunn's Index With clusters that are **loose** or **somewhat similar to one another**, we expect the DBI to be **large**. There are other ways to assess separation and tightness; **Dunn's index** is the ratio of the **smallest between-cluster dissimilarity** (for pairs of observations not in the same cluster) to the **largest cluster diameter** (within-cluster dissimilarity).



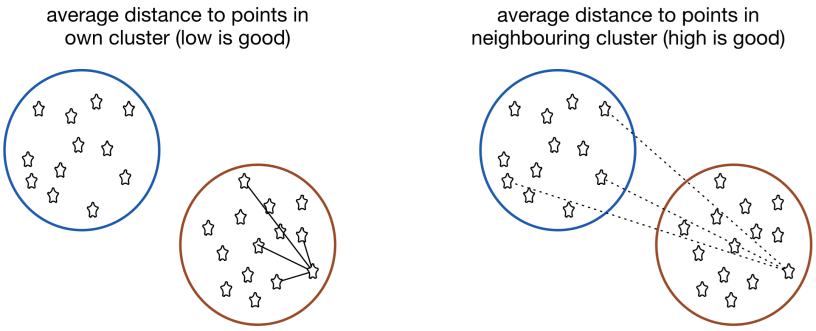
If the clusters $\{C_k\}$ are **tight** and **dissimilar to one another**, we expect the smallest between-cluster dissimilarity to be large and the largest cluster diameter to be small, leasing to a **large** Dunn ratio.

Conversely, with clusters that are **loose** or **somewhat similar to one another**, the Dunn ratio will be **small**. As is the case with the sum of errors and the DBI, the choice of the dissimilarity (or distance) measure leads to different variants of the Dunn index, but all of them take non-negative values.

Silhouette Metric The three previous CQMs provide examples of validation metrics that are computed for the entire clustering outcome; the **silhouette metric** instead assigns a score to each observation, and aggregates the scores at a cluster level and at the dataset level: for a dissimilarity d and a point $\mathbf{x}$ in a cluster C, set

$$b(\mathbf{x}) = \min_{C' \neq C} \left\{ \text{Avg}_{\mathbf{y} \in C'} \{d(\mathbf{x}, \mathbf{y})\} \right\}, \quad a(\mathbf{x}) = \text{Avg}_{\substack{\mathbf{w} \in C \\ \mathbf{w} \neq \mathbf{x}}} \{d(\mathbf{x}, \mathbf{w})\}.$$

Small values of $a(\mathbf{x})$ imply that $\mathbf{x}$ is similar to the instances in its cluster, large values of $b(\mathbf{x})$ imply that it is dissimilar to instances in other clusters.



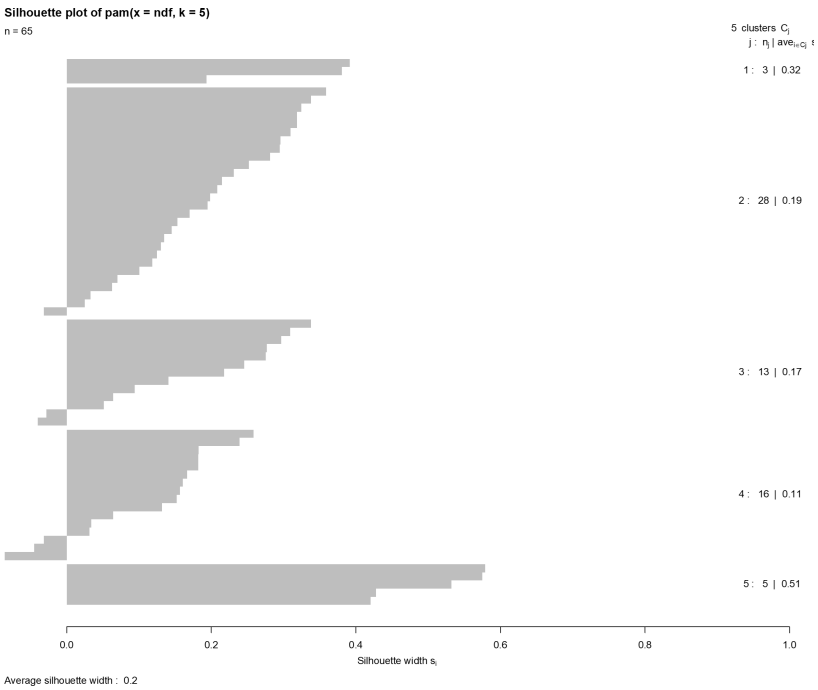
The silhouette metric at $\mathbf{x}$ is

$$\text{silhouette}(\mathbf{x}) = \frac{b(\mathbf{x}) - a(\mathbf{x})}{\max\{a(\mathbf{x}), b(\mathbf{x})\}} = \begin{cases} 1 - a(\mathbf{x})/b(\mathbf{x}) & \text{if } a(\mathbf{x}) < b(\mathbf{x}) \\ 0 & \text{if } a(\mathbf{x}) = b(\mathbf{x}) \\ b(\mathbf{x})/a(\mathbf{x}) - 1 & \text{if } a(\mathbf{x}) > b(\mathbf{x}) \end{cases}$$

Consequently, $-1 \leq \text{silhouette}(\mathbf{x}) \leq 1$ for all $\mathbf{x}$. Thus, when silhouette($\mathbf{x}$) is large (≈ 1), the ratio $a(\mathbf{x})/b(\mathbf{x})$ is small and we interpret $\mathbf{x}$ to be correctly assigned to cluster C (and conversely, when silhouette($\mathbf{x}$) is small (≈ -1), we interpret $\mathbf{x}$'s assignment to C to be "incorrect").

The **silhouette diagram** corresponding to the clustering results displays the silhouette scores for each observation, and averages out the scores in each cluster. The mean over all observations (and the number of instances that have been poorly assigned to a cluster) gives an indication of the appropriateness of the clustering outcome.

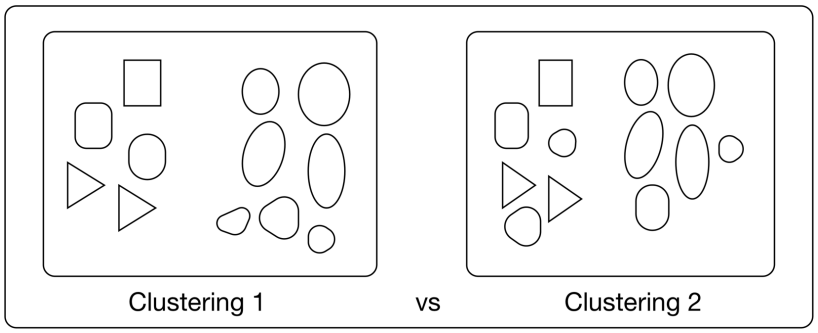
In the example below, 65 observations are separated into 5 clusters: 6 observations are **poorly assigned** (those with negative silhouette scores) and the **average silhouette score** over the entire dataset is 0.2, which suggests that the clustering is more successful than unsuccessful, overall.



Note, however that it could prove difficult to associate intrinsic meaning to a **lone numerical score** – there could be contexts where 0.2 is considered to be a fantastic silhouette score, and others where it is viewed as an abject failure. It is in comparison to the scores obtained by different clustering schemes that this score (and those of the other CQMs) can best serve as an **indicator** of a strong (or a poor) clustering outcome.

Relative Validation Obtaining a **single validation measure** for a **single clustering outcome** is not always that useful – how would we really characterize the silhouette score of 0.2 in the previous example? Could the results be better? Is this the best that can be achieved?

One approach is to compare clustering outcomes across multiple runs to take advantage of non-deterministic algorithms or various parameters' values (see image below, and schematics in Figure 22.10).



If the outcomes of different clustering algorithms on the same dataset are "similar", this provides evidence in favour of the resulting clusters being **efficient**, **natural**, or **valid**, in some sense.

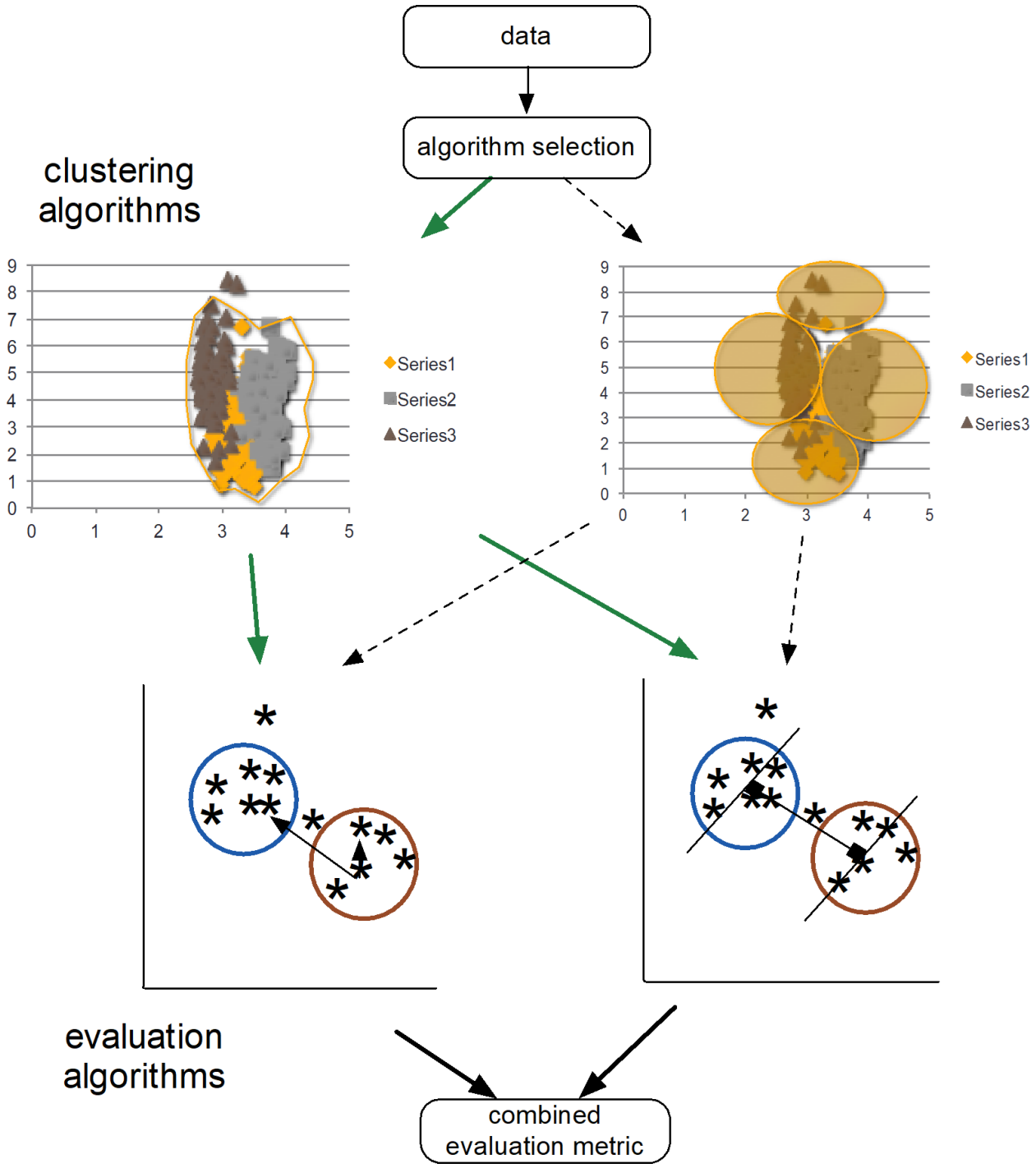
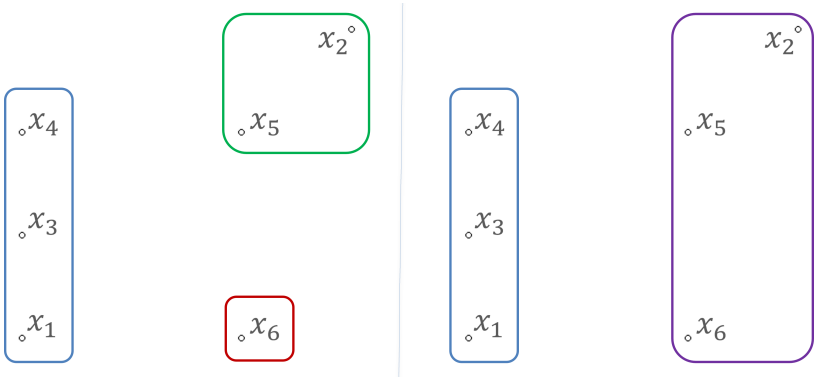


Figure 22.10: Schematics of relative clustering validation.

Consider, for instance, a dataset with 6 instances, which is clustered in two different manners ($\mathscr{A}$ and $\mathscr{B}$, with $|\mathscr{A}| = 3$ and $|\mathscr{B}| = 2$), as shown below. The clusterings are clearly not identical; how similar are they?



One way to approach relative validation for two outcomes is by computing "**correlations**" between cluster outcomes. Intuitively, we would expect the similarity between both clustering schemes to be high, seeing as the two outcomes are not that different from one another.[28]

28: In $\mathscr{B}$, the two smallest clusters of $\mathscr{A}$ have been merged into a single, larger cluster.

This can be represented in the form of matrices:

	P1	P2	P3	P4	P5	P6
P1	1					
P2	0	1				
P3	1	0	1			
P4	1	0	1	1		
P5	0	1	0	0	1	
P6	0	0	0	0	0	1

	P1	P2	P3	P4	P5	P6
P1	1					
P2	0	1				
P3	1	0	1			
P4	1	0	1	1		
P5	0	1	0	0	1	
P6	0	1	0	0	1	1

How can this be quantified? **Correlation measures** include the Rand, Jaccard (see Chapter 26), and Gamma (see [51]) indices.

Let $\mathscr{A} = \{A_1, \ldots, A_k\}$ and $\mathscr{B} = \{B_1, \ldots, B_\ell\}$ be two clusterings of a dataset $\mathbf{X}$. If $\mathbf{X}$ consists of n instances, there are thus

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

pairs of observations in the data. Denote the number of pairs of observations that are in:

- the same cluster in $\mathscr{A}$ **and** the same cluster in $\mathscr{B}$ by ss,
- different clusters in $\mathscr{A}$ **and** different clusters in $\mathscr{B}$ by dd;
- the same cluster in $\mathscr{A}$ **but** different clusters in $\mathscr{B}$ by sd, and
- different clusters in $\mathscr{A}$ **but** the same cluster in $\mathscr{B}$ by ds.

Thus,

$$\binom{n}{2} = \text{ss} + \text{dd} + \text{sd} + \text{ds},$$

and we define the **Rand index** of $\mathscr{A}$ and $\mathscr{B}$ as

$$\text{RI}(\mathscr{A}, \mathscr{B}) = \frac{\text{ss} + \text{dd}}{\text{ss} + \text{dd} + \text{sd} + \text{ds}} = \frac{\text{ss} + \text{dd}}{\binom{n}{2}}.$$

Large values of the index are indicative of similarity of clustering outcomes.[29] Unfortunately, the Rand index does not satisfy the **constant baseline property**.[30]

29: The formula for $\text{RI}(\mathscr{A}, \mathscr{B})$ reminds one of the definition of accuracy, a performance evaluation measure for (binary) classifiers.

30: In a nutshell, the expected value of $\text{RI}(\mathscr{A}, \mathscr{B})$ for independent, random clusterings $\mathscr{A}$ and $\mathscr{B}$ is not 0 [45].

The **adjusted Rand index** (as well as several other pair-counting, set-matching, and information theoretic measures) relies on the **contingency table** between $\mathscr{A}$ and $\mathscr{B}$, a method to summarize the outcomes of two clustering results on the same dataset:

	B_1	$\cdots$	B_ℓ	Total
A_1	$n_{1,1}$	$\cdots$	$n_{1,\ell}$	μ_1
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
A_k	$n_{k,1}$	$\cdots$	$n_{k,\ell}$	μ_k
Total	ν_1	$\cdots$	ν_ℓ	n

In this array, $n_{i,j} = |A_i \cap B_j|$ represents the number of instances that are found in **both** the cluster $A_i \in \mathscr{A}$ and $B_j \in \mathscr{B}$. The adjusted Rand index ($\in [-1, 1]$) is given by

$$\text{ARI}(\mathscr{A}, \mathscr{B}) = \frac{\sum_{ij} \binom{n_{ij}}{2} - \left[\sum_i \binom{\mu_i}{2} \sum_j \binom{\nu_j}{2}\right] \Big/ \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{\mu_i}{2} + \sum_j \binom{\nu_j}{2}\right] - \left[\sum_i \binom{\mu_i}{2} \sum_j \binom{\nu_j}{2}\right] \Big/ \binom{n}{2}},$$

which can also be re-written as

$$\text{ARI}(\mathscr{A}, \mathscr{B}) = \frac{2(\text{ss} \cdot \text{dd} - \text{sd} \cdot \text{ds})}{(\text{ss} + \text{sd})(\text{ds} + \text{dd}) + (\text{ss} + \text{ds})(\text{sd} + \text{dd})}.$$

It is straightforward to compute RI and ARI for the two clusterings of the artificial example with 6 instances. Since $n = 6$, there are $6 \cdot 5/2 = 15$ pairs of observations in the data, and we have:

- ss = 4 (x_1 and x_3; x_1 and x_4; x_3 and x_4; x_2 and x_5);
- ds = 2 (x_2 and x_6; x_5 and x_6);
- sd = 0 (none);
- dd = 9 (all remaining pairs).

Thus,

$$\text{RI}(\mathscr{A}, \mathscr{B}) = \frac{4 + 9}{4 + 9 + 0 + 2} = \frac{13}{15} = 0.87$$
$$\text{ARI}(\mathscr{A}, \mathscr{B}) = \frac{2(4 \cdot 9 - 0 \cdot 2)}{(4 + 0)(2 + 9) + (4 + 2)(0 + 9)} = 0.73.$$

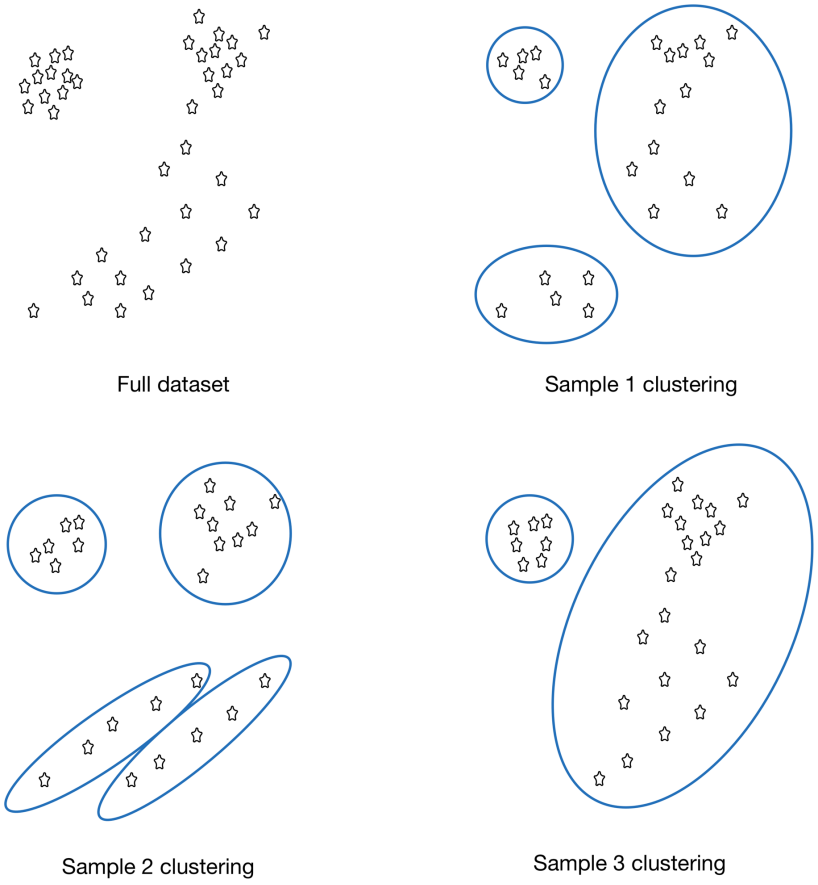
Both of these values are indicative of high similarity between the clustering outcomes $\mathscr{A}$ and $\mathscr{B}$.

Cluster **stability** can also be used to assess the appropriateness of the choice of algorithm for the data. Assume that we apply a clustering algorithm $\mathcal{G}$ to a dataset $\mathbf{X}$, resulting in a clustering scheme $\mathcal{C} = \{C_1, \ldots, C_K\}$.

In general, a dataset $\mathbf{X}$ is a **realization** of a (potentially unknown) underlying data generating mechanism related to the situation of interest; a different realization $\mathbf{X}'$, which could arise from the collection of new data, may yield a distinct clustering scheme $\mathcal{C}'$. How **stable** is the clustering outcome of $\mathcal{G}$, relative to the realization $\mathbf{X}$?

We can get a sense for the underlying stability by sampling multiple row subsets from $\mathbf{X}$;[31] however this is achieved, we have obtained ℓ subsets $\mathbf{X}_1, \ldots, \mathbf{X}_\ell \subseteq \mathbf{X}$, on which we apply the algorithm $\mathcal{G}$, with parameters $\mathcal{P}$, yielding the corresponding clustering schemes $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$.

31: Alternatively, we could also sample from $\mathbf{X}$'s columns, or sample columns from a variety of sampled rows of $\mathbf{X}$.



Let $\mathcal{W}$ be the similarity matrix for the various clustering schemes:

$$\mathcal{W} = \begin{pmatrix} w(\mathcal{C}_1, \mathcal{C}_1) & \cdots & (\mathcal{C}_1, \mathcal{C}_\ell) \\ \vdots & \ddots & \vdots \\ w(\mathcal{C}_\ell, \mathcal{C}_1) & \cdots & (\mathcal{C}_\ell, \mathcal{C}_\ell) \end{pmatrix};$$

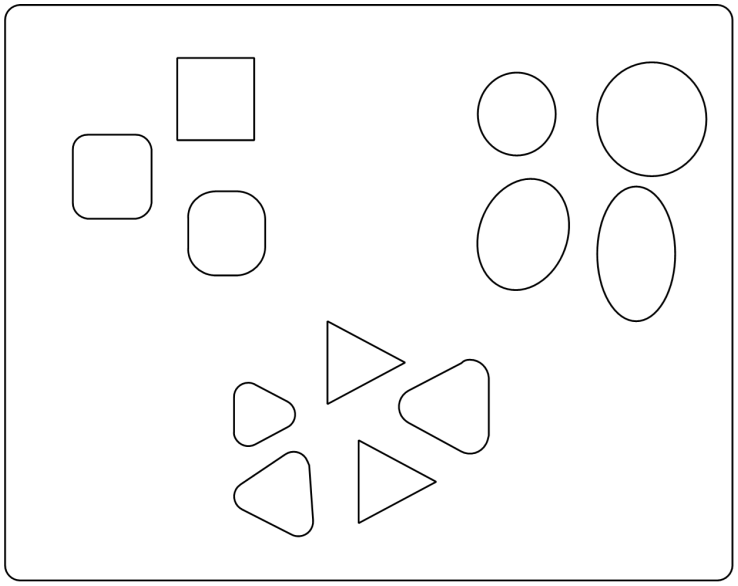
this $\mathcal{W}$ can be used as the basis of a **meta-clustering scheme** in which $\mathcal{C}_1, \ldots, \mathcal{C}_\ell$ play the role of instances, using a graph-based meta-clustering method such as DBSCAN (which we will discuss in Section 22.4.1). If the clustering results obtained from $\mathbf{X}$ by applying $\mathcal{G}$ are **stable**, we might expect the meta-clustering results to yield a **single meta-cluster**.[32]

32: If multiple meta-clusters are found from $\mathcal{W}$, this supports the position that $\mathcal{G}$ does not produce stable clusters in $\mathbf{X}$, although this does not necessarily imply **instability** as the number of meta-clusters could be an artefact related to the choice of the meta-clustering method. This approach seems simple in theory, but in practice it often simply pushes the issues and challenges of clustering to the meta-clustering activity; a more sophisticated treatment of the problem of cluster stability assessment is presented in [27].

External Validation In everyday applications, we tend to gauge clustering results against some (often implicit) **external expectation**: we cannot help but bring in outside information to evaluate the clusters.

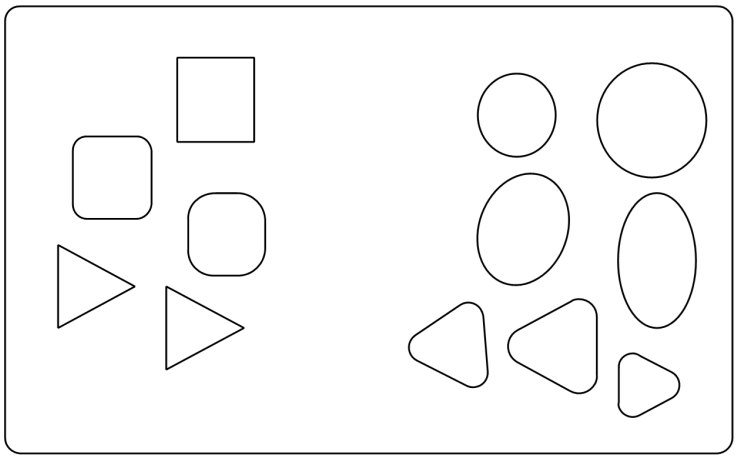
The outside information is typically what is deemed to be the 'correct' groups to which the instances belong.[33] In the example below, for instance, we might be interested in finding **natural groups** in a dataset of objects, but we might hold the pre-conceived notion that the natural groups are linked to the underlying **shape** (square, triangle, circle).

33: This is starting to look a lot like **classification**, a supervised learning approach.



Natural Groupings

If the outcome agrees with this (external) notion, we naturally feel that the clustering was successful; if, the outcome seems to pick up something about the sharpness of the shapes' vertices, say (as in the image below), we might conclude that the algorithm does not do a good job of capturing the essential nature of the objects, or, more rarely, that we need to revisit our pre-conceived notions about the dataset and its natural groups.



Clustering Results

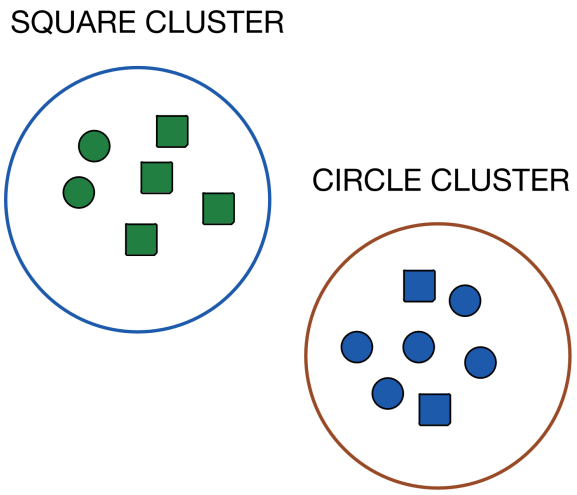
This validation strategy is often used to **build confidence** in the overall approach, based on preliminary or sample results, but it rests on a huge assumption (which often conflicts with the unsupervised learning framework), namely, that the natural groups in the data are **identifiable**, explicitly or implicitly.

34: Which it is emphatically not, it bears repeating.

Due to the conceptual similarity to classification,[34] **external validation measures** often resemble classification performance measures. Case in point, the **purity metric**. In the presence of an external classification variable, this metric assign a label to a cluster C according to the most frequent classes of the instances in C, and

$$\text{purity}(C) = \frac{\#\text{ correctly assigned points in } C}{|C|},$$

as in the example below:



The **clustering purity score** for $\mathscr{C} = \{C_1, \ldots, C_K\}$ is obtained as the average of the cluster purity scores, or as a weighted average of purity scores:

$$\text{weighted purity}(\mathscr{C}) = \frac{1}{n}\sum_{\ell=1}^{K} |C_\ell| \cdot \text{purity}(C_\ell),$$

where n represents the number of instances in the data.

In the image above, the green cluster is labeled as the square cluster (since 4 of its 6 instances are classified as squares), and the blue cluster is labeled as the circle cluster (since 5 of its 7 instances are classified as circles). At the cluster level, the purity scores are thus:

$$\text{purity}(C_\square) = \frac{2}{3}, \quad \text{purity}(C_\bigcirc) = \frac{5}{7};$$

the average and weighted purity scores are

$$\text{average purity}(\mathscr{C}) = \frac{1}{2}\left(\frac{2}{3} + \frac{5}{7}\right) = 69.0\%$$

$$\text{weighted purity}(\mathscr{C}) = \frac{1}{6+7}\left(6 \cdot \frac{2}{3} + 7 \cdot \frac{5}{7}\right) = 69.2\%.$$

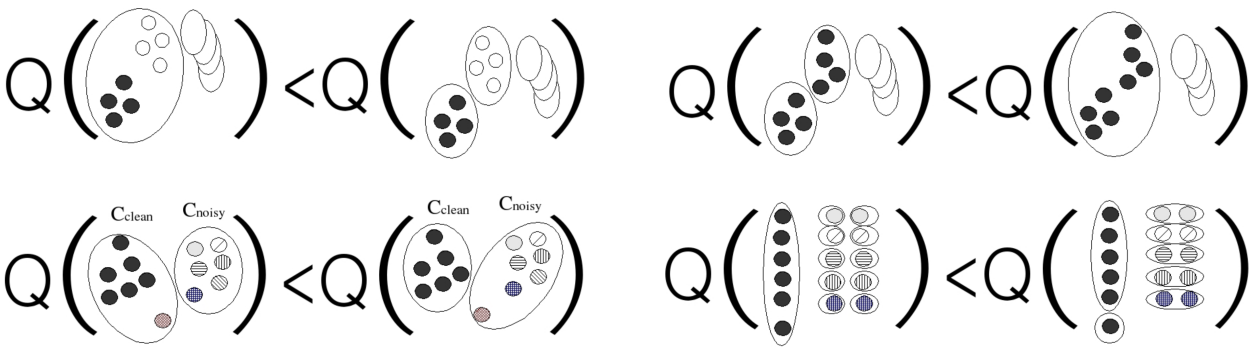


Figure 22.11: Useful external quality metric considerations: homogeneity (top left), completeness (top right), noisy and outlying data (bottom left), size and quantity (bottom right).

These two numbers are very nearly identical because the clusters have roughly the same size; if the size variance is large, the two measurements would be quite different. The purity is an obvious analogue to **accuracy**; other measures based on **precision** and **recall** are also popular [3].

Useful external quality metrics take advantage of the natural classes (if they are aligned with the clustering results), and take into account cluster **homogeneity** (top left, Figure 22.11), **completeness**, (top right), **noisy and outlying data** (bottom left), and **size vs. quantity** considerations (bottom right): the preferred behaviour is shown on the right [3].

Example Let us illustrate some of these notions using various $k-$means and hierarchical clusters of the Gapminder data used in the previous sections. In all instances, we use Euclidean distance on the scaled dataset.

Internal Validation We use the `R` packages `cluster`, `fpc`, and `clusterCrit` to compute 3 CQMs: the Dunn index, the average silhouette width, and the **Calinski-Harabasz** index, which is simply the ratio of the sum of **between-clusters dispersion** to the **inter-cluster dispersion** for all clusters (higher is better).

We start by clustering the data using 4−means; we then use hierarchical clustering with complete linkage and 3 clusters (the global structure has already been computed).

```
set.seed(123) # for replicability
kmeans.4 = kmeans(gapminder.SoCL.2011.s,4,iter.max=2509,nstart=1)
stats.kmeans.4 <- as.numeric(
    clusterCrit::intCriteria(as.matrix(gapminder.SoCL.2011.s),
    kmeans.4$cluster,
    c("Dunn","Silhouette","Calinski_Harabasz")))

dist.all <- cluster::daisy(gapminder.SoCL.2011.s,metric="euclidean",stand=FALSE)
hc.1.3 <- cutree(hclust.gapminder.SoCL.2011, k = 3)
stats.hc.1.3 <- c(fpc::cluster.stats(dist.all, clustering=hc.1.3, silhouette = TRUE)$dunn,
    fpc::cluster.stats(dist.all, clustering=hc.1.3, silhouette = TRUE)$avg.silwidth,
    fpc::cluster.stats(dist.all, clustering=hc.1.3, silhouette = TRUE)$ch)
```

The results are summarized below:

```
stats <-rbind(stats.kmeans.4,stats.hc.1.3)
colnames(stats) <- c("Dunn",
  "Silhouette",
  "Calinski-Harabasz")
stats
```

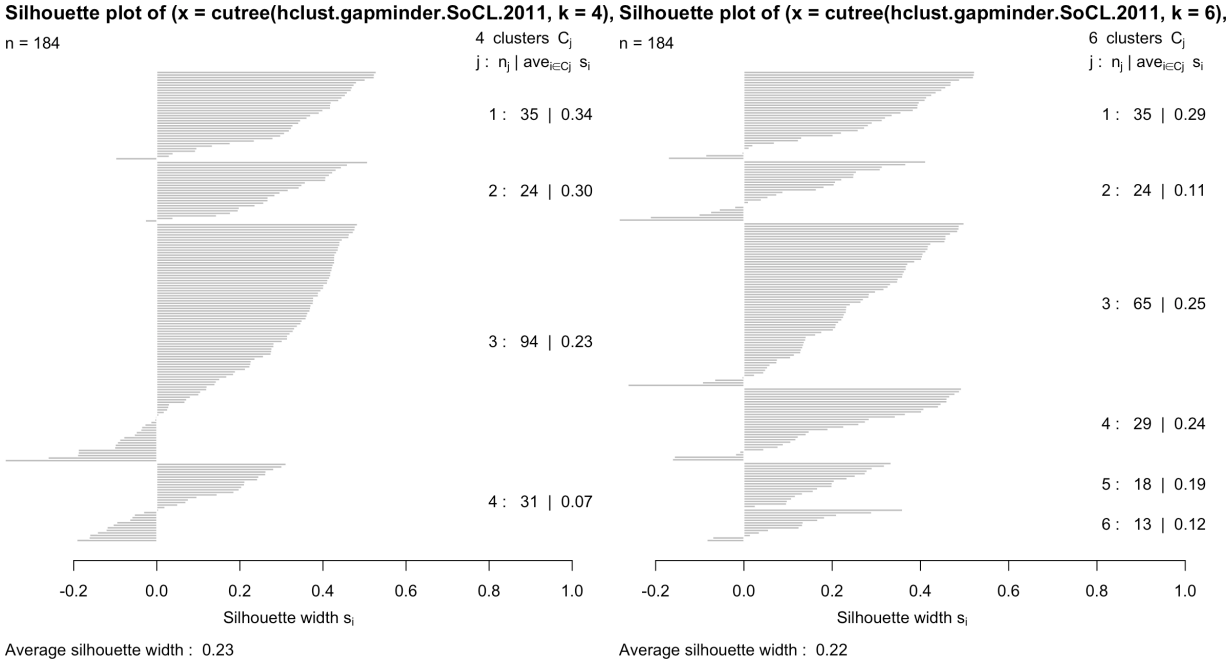
method	Dunn	Avg. Sil.	C.-H.
4−means	0.097	0.315	139.0
HC(comp; 3)	0.091	0.274	125.4

Both of the Dunn values are low, although the 4−means result is marginally superior; while the average silhouette widths are also low, they are least positive in both cases, with a slight advantage in favour of 4−means; the Calinski-Harabasz values are not very indicative on their own, but once again, 4−means comes out ahead of HC.

The average silhouette width is an intriguing metric. On the one hand, we attempt to gauge the quality of the **entire clustering** with a single number, but the average is a fickle summary measurement and potentially affected by outlying values; on the other, we do have access to a silhouette score **for each observation**, and can thus get a better idea of the performance by studying the **silhouette profile**.

We show the silhouette results for hierarchical clustering with complete linkage for 4 (average width= 0.23) and 6 clusters (average width= 0.22).

```
plot(cluster::silhouette(cutree(hclust.gapminder.SoCL.2011, k = 4), dist.all))
plot(cluster::silhouette(cutree(hclust.gapminder.SoCL.2011, k = 6), dist.all))
```



The average silhouette width seems to favour the 4-cluster result, the profile for the 6-cluster result seems more in-line with desirable properties: in both instances, some observations are "mis-clustered" (negative silhouette scores), but these seem to be more broadly distributed in the latter case.[35]

35: in the 4-cluster case, half a cluster seems to have been mis-assigned, for instance.

Relative Validation We compute the Rand index (RI) and the adjusted Rand index (ARI) to compare the outcomes of a single run of 2−means ($\mathscr{A}_2$), 3−means ($\mathscr{A}_3$), and 4−means ($\mathscr{A}_4$), respectively.

```
set.seed(1) # for replicability
kmeans.2 = kmeans(gapminder.SoCL.2011.s,2,iter.max=250,nstart=1)
kmeans.3 = kmeans(gapminder.SoCL.2011.s,3,iter.max=250,nstart=1)
kmeans.4 = kmeans(gapminder.SoCL.2011.s,4,iter.max=250,nstart=1)
```

We can compute the Rand index and the adjusted Rand index using the following function:

```
# create a matrix of 1s and 0s depending as to whether
# observations i and j are in the same cluster or not
w2=kmeans.2$cluster
mat2=floor(1 - abs(sqrt(w2%*%t(w2))) %% 1)

w3=kmeans.3$cluster
mat3=floor(1 - abs(sqrt(w3%*%t(w3))) %% 1)

w4=kmeans.4$cluster
mat4=floor(1 - abs(sqrt(w4%*%t(w4))) %% 1)

# build the rand index from these matrices
randindices  <- function(W1,W2) {
  diag(W1) <- -1
  diag(W2) <- -1
  W=table(W1+2*W2)
  W=W[-c(1)]
  dd=W[1]
  sd=W[2]
  ds=W[3]
  ss=W[4]
  RI=(ss+dd)/(ss+dd+sd+ds)
  ARI=2*(ss*dd-sd*ds)/((ss+sd)*(ds+dd)+(ss+ds)*(sd+dd))
  randindices=data.frame(cbind(RI[[1]],ARI[[1]]))
}

# compute RI and ARI for the 3 comparisons
(randindices(mat2,mat3))
(randindices(mat2,mat4))
(randindices(mat3,mat4))
```

```
    RI    ARI       RI    ARI       RI    ARI
0.7327 0.5152   0.6407 0.3285   0.7549 0.4584
```

There are $\binom{184}{2} = 16836$ pairs of distinct observations in the 2011 Gapminder dataset; the full pair types and indices break down as below:

Schemes	ss	dd	sd	ds	RI	ARI
$\mathscr{A}_2, \mathscr{A}_3$	5304	7032	3852	648	0.73	0.52
$\mathscr{A}_2, \mathscr{A}_4$	4395	6392	4761	1288	0.64	0.33
$\mathscr{A}_3, \mathscr{A}_4$	3754	8955	2198	1929	0.75	0.46

$\mathscr{A}_2, \mathscr{A}_3$ are relatively similar according to RI, as are $\mathscr{A}_3, \mathscr{A}_4$, but the ARI suggests that $\mathscr{A}_2, \mathscr{A}_3$ are more similar to one another than $\mathscr{A}_3, \mathscr{A}_4$ are; $\mathscr{A}_2, \mathscr{A}_4$ are not as similar, according to both indices, which is not that surprising as the number of clusters in this case jumps from 2 to 4.

External Validation Finally, we compare the clustering results of hierarchical clustering, for 4 and 8 clusters, with a variety of external grouping: 6 world regions, as determined by the Gapminder Foundation, and OECD/G77 membership (see Figure 22.12).

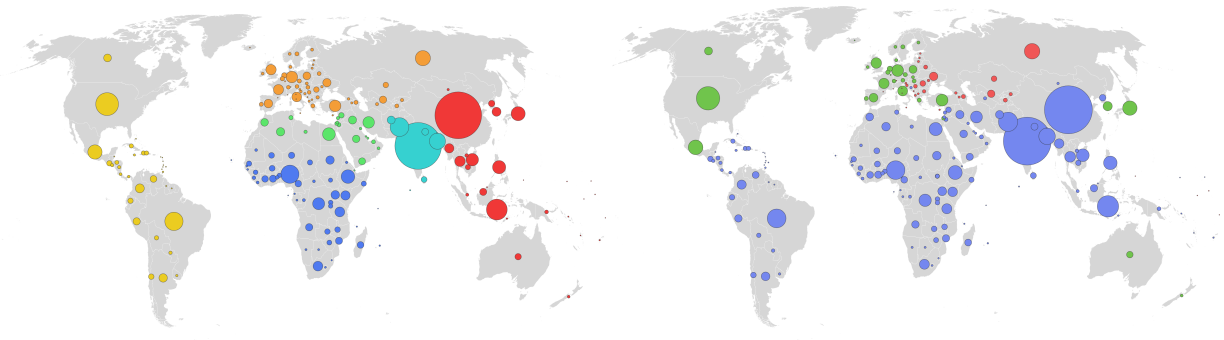


Figure 22.12: 6 world regions (left): America (yellow, 33 countries), East Asia Pacific (red, 26), Europe Central Asia (orange, 49), Middle East North Africa (green, 20), South Asia (turquoise, 8), Sub Saharan Africa (blue, 48); memberships (right): OECD (green, 30), G77 (purple, 128), other (red, 26); bubble size represents population [36].

We start by importing the external groupings.

```
gapminder.regions = read.csv("gapminder_regions.csv",
    stringsAsFactors=TRUE)
colnames(gapminder.regions)[1] <- c("geo")
```

Then, we cluster the data using HC with complete linkage, for $k = 4$ and $k = 8$ clusters (using Euclidean dissimilarity). Recall that the dendrogram structure was originally stored in `hclust.gapminder.SoCL.2011`.

```
hc.4.ce <- as.factor(cutree(hclust.gapminder.SoCL.2011, k = 4)) # complete, Euclidean
hc.8.ce <- as.factor(cutree(hclust.gapminder.SoCL.2011, k = 8)) # complete, Euclidean

geo = names(hc.4.ce)
clusters=data.frame(geo,hc.4.ce,hc.8.ce)
external.results <- merge(gapminder.regions,clusters, by="geo")
```

The first 3 entries of the `external.results` are shown on the next page.

```
head(external.results,3)
```

```
geo name         four_regions  eight_regions       six_regions          members_oecd_g77  hc.4.ce  hc.8.ce
afg Afghanistan  asia          asia_west           south_asia           g77               1        1
ago Angola       africa        africa_sub_saharan  sub_saharan_africa   g77               1        1
alb Albania      europe        europe_east         europe_central_asia  others            3        3
```

The clusters are then labeled with their most frequent cluster assignment, which can be extracted with the following function:

```
mode <- function(x) { names(which.max(table(x))) }

tab <- external.results |>  group_by(hc.4.ce) |>
  summarise(mode = mode(six_regions), n=n())
n.mode <- external.results |>  group_by(hc.4.ce) |>
  count(six_regions) |>
  summarise(n.mode = max(n))
info <- merge(tab,n.mode, by="hc.4.ce")[,2:4]
```

Are there any reasons to suspect that the clusters would be aligned with these external classes? For the 6 world regions classes, the clusters labels (the most frequent class per cluster) for HC(4) are shown below:

Cluster	Label	Size	Frequency
1	Sub Saharan Africa	35	31
2	East Asia Pacific	24	9
3	Europe Central Asia	94	42
4	Sub Saharan Africa	31	14

```
info[,4] <- info$n.mode/info$n
(purity <- mean(info$V4))
(weighted.purity <- weighted.mean(info$V4,info$n))
```

This clustering scheme yields a purity of 0.54 and a weighted purity of 0.52 – the overall score is not great, but the Sub Saharan countries are fairly well captured by clusters 1 and 4.

We repeat the same process for HC(8) (the code is omitted). The clusters labels in that case are found below:

Cluster	Label	Size	Frequency
1	Sub Saharan Africa	35	31
2	America	17	6
3	Europe Central Asia	65	34
4	East Asia Pacific	7	3
5	america	29	10
6	Sub Saharan Africa	18	7
7	East Asia Pacific	10	5
8	Sub Saharan Africa	3	3

This now yields a purity of 0.55 and a weighted purity of 0.54; which is still .... not that great. Perhaps the clusters have little to do with geography, in the end.

Are they aligned with OECD/G77/other membership? The labels for HC(8) with this external grouping are found below:

Cluster	Label	Size	Frequency
1	G77	27	27
2	G77	29	22
3	OECD	28	17
4	G77	20	18
5	G77	12	11
6	OECD	23	11
7	G77	25	24
8	G77	20	10

The purity values are 0.77 and 0.76, respectively: these are better values than the previous external validation attempts, but they might not really be meaningful given the preponderance of G77 countries in the data.

It seems, then, that neither of the external classifications is a good gauge of cluster validity for this data.

For the most part, the cluster validation yields middling results. The few algorithms we have tried with the data suggest that there is some low-level grouping at play, but nothing we have seen so far would suggest that the data segments are all that "natural."

While this result is somewhat disappointing, we should note that this is often the case with real-world data: there is no guarantee that natural groups even exist in the data. However, we have not been directing our choices of algorithms and parameters – up to now, they have been made fairly arbitrary. Can anything be done to help with **model selection**?

22.3.2 Model Selection

How do we pick the number of clusters and the various other parameters (including choice of algorithm) to use for "optimal" results? A common approach is to look at all the outcomes obtained from various parameter runs and replicates (for a given algorithm), and to select the parameter values that optimize a set of QCMs, such as Davies-Bouldin, Dunn, CH, etc.

Optimization is, of course, dependent on each QCM's properties: in some cases, we are searching for parameters that maximizes the index, in other cases, those that minimize it, and yet in other cases, for "knees" or "change points" in various associated plots.

Note, however, that the parameter values that optimize a QCM may not optimize others; when they coincide, this reinforces the support for the existence of natural groups; when they do not, they provide a smaller collection of models from which to select, removing some of the arbitrariness discussed above.

This can also be done for clustering outcomes arising from different algorithms, although in this case we are not selecting parameter values so much as identifying the model that best describes the natural groups in the data among all results, according to some metric(s).

The metrics presented in Section 22.3.1 all provide frameworks to achieve this. There are additional approaches, such as: seeking the clustering $\mathscr{C} = \{C_1, \ldots, C_k\}$, among a list of such outcomes, which minimizes the **quadratic cost**

$$\Lambda_{\mathbf{W}}(\mathscr{C}) = -\text{trace}\left(Z^{\top}(\mathscr{C})\mathbf{W}Z(\mathscr{C})\right),$$

where $z_{i,\ell} = 1$ is $\mathbf{x}_i \in C_\ell$ and 0 otherwise, associated with a **similarity matrix W**; or methods relying on stability assessment [27, 25]. Model assessment and selection remains a popular research topic.

But it remains important to remember that there is a lot of diversity in clustering validation techniques. The various types of validation methods do not always give concordant results; this variation within the types can be demoralizing at times, but it can also be leveraged to extract useful information about the data's **underlying structure**.

In general, we should avoid using a single assessment method; it is preferable to seek "signals of agreement" across a **variety of strategies** (both in the choices of clustering algorithms and evaluation methods). Finally, remember that clustering results may just be 'ok' ... but that is ok too! We can study the situation and decide what is important and what can safely be ignored – as always, **a lot depends on the context**.

Example How many clusters k should we seek when clustering the (scaled) 2011 Gapminder dataset using Euclidean distance? For each $k = 2, \ldots, 15$, we compute the outcome of $m = 40$ runs of $k-$means, and average the **within sum of squares** (WSS) and a (modified) **Davies-Bouldin** index (DBI) over the runs. The optimal number of parameters is obtained at a DBI maximum or a WSS "knee".

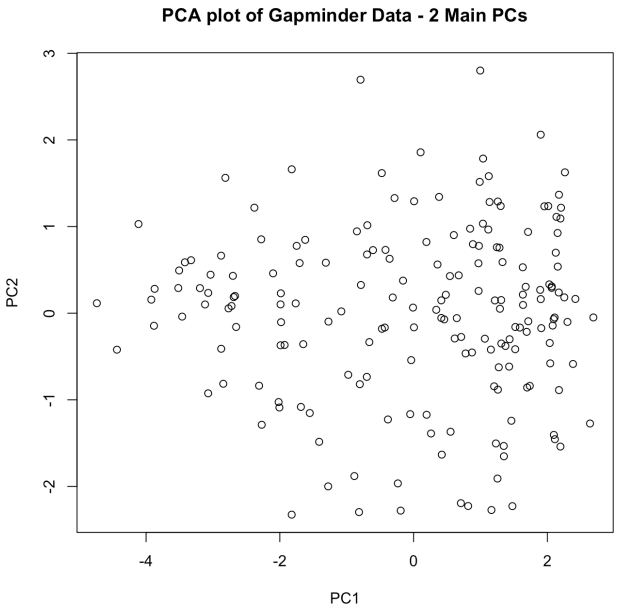
We start by computing the principal components for displaying purposes – although we could also use them to cluster the data, at the cost of some information about the dataset.

Principal component distribution

```
decomposition
pc.agg.data = princomp(gapminder.SoCL.2011.s)
summary(pc.agg.data)
```

```
Importance of components:
                          Comp.1    Comp.2     Comp.3     Comp.4    Comp.5
Standard deviation     1.850624 0.9973737 0.49950729 0.45130387 0.3163521
Proportion of Variance 0.688705 0.2000380 0.05017419 0.04095763 0.0201251
Cumulative Proportion  0.688705 0.8887431 0.93891726 0.97987490 1.0000000
```

```
pc.df.agg.data = cbind(pc.agg.data$scores[,1],
    pc.agg.data$scores[,2])
plot(pc.df.agg.data, xlab="PC1", ylab="PC2")
title('PCA plot of Gapminder Data - 2 Main PCs')
```

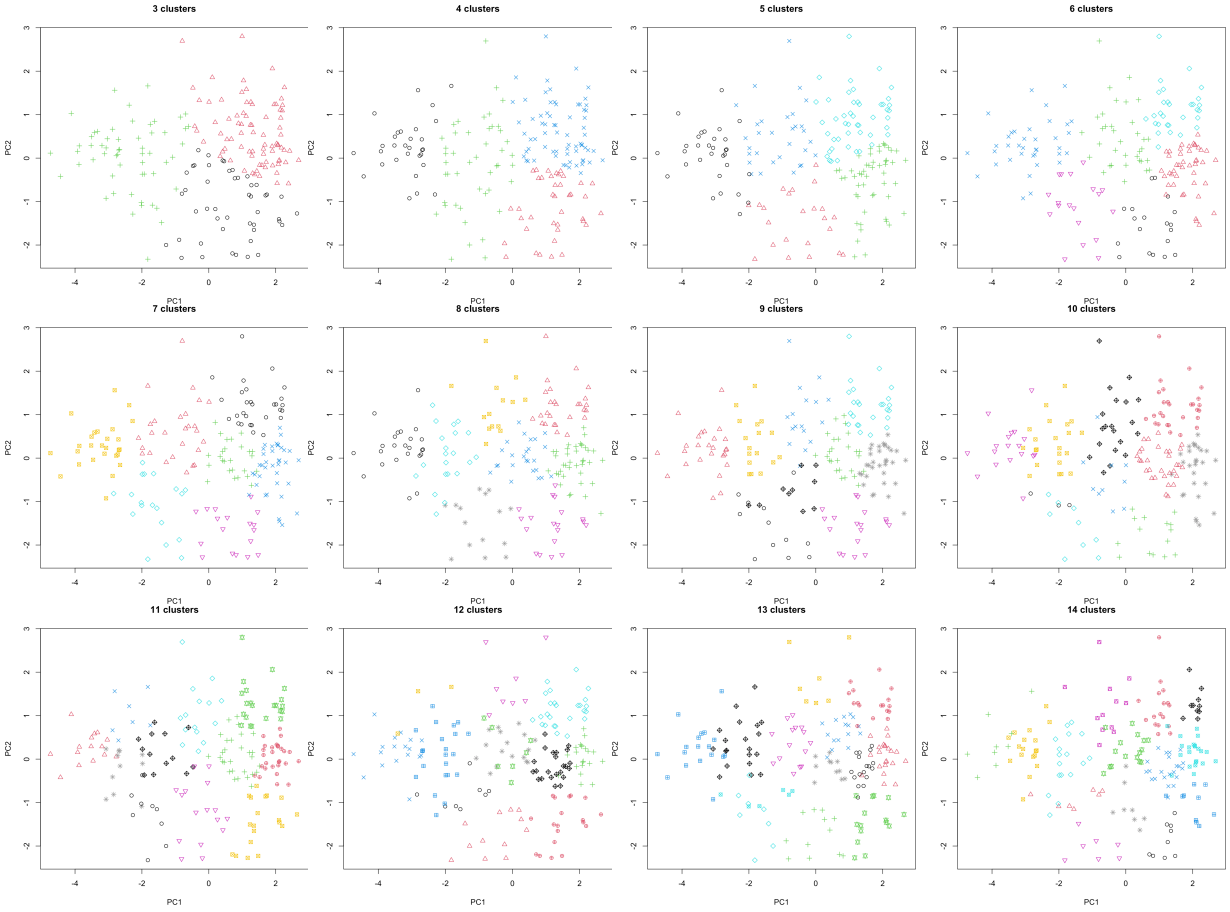


The Davies-Bouldin index is computed using the following formula.

```
Davies.Bouldin <- function(A, SS, m) {
  # A  - the centres of the clusters
  # SS - the within sum of squares
  # m  - the sizes of the clusters
  N <- nrow(A)   # number of clusters
  # intercluster distance
  S <- sqrt(SS/m)
  # Get the distances between centres
  M <- as.matrix(dist(A))
  # Get the ratio of intercluster/centre.dist
  R <- matrix(0, N, N)
  for (i in 1:(N-1)) {
    for (j in (i+1):N) {
      R[i,j] <- (S[i] + S[j])/M[i,j]
      R[j,i] <- R[i,j]
    }
  }
  return(mean(apply(R, 1, max)))
}
```

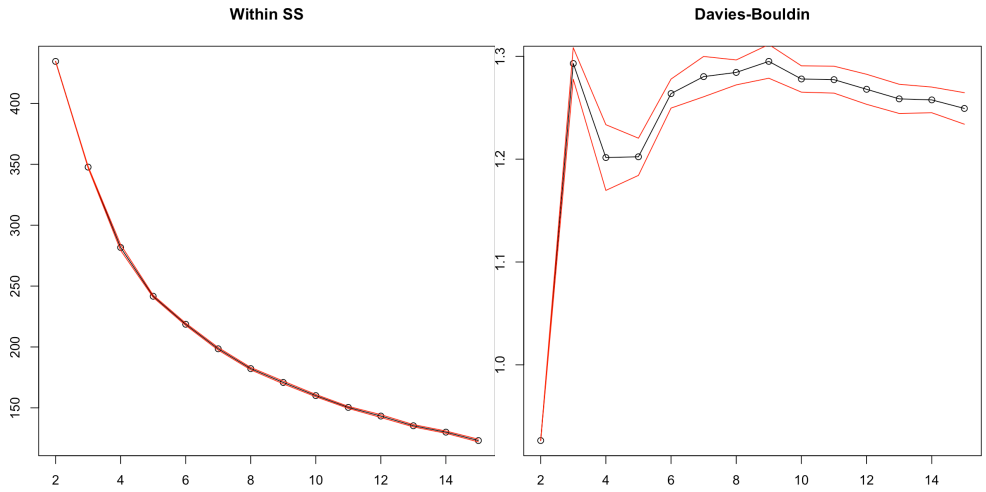
For each $k = 2, \ldots, 15$, we run $k-$means $N = 40$ times (using Euclidean dissimilarity). One realization is displayed for each k, as are the DBI curves and the WSS curves (with confidence bands). The clusters are displayed on the first 2 principal components of the dataset, which explain 88% of the variation in the data.

```
N = 40            # Number of repetitions
max.cluster = 15  # Number of maximum number of desired clusters

# initializing values
m.errs = m.DBI = s.errs = s.DBI <- rep(0, max.cluster)

# clustering and plotting
set.seed(1)
for (i in 2:max.cluster) {
  errs = DBI <- rep(0, max.cluster)

  for (j in 1:N) {
    # data, number of internal shifts of the cluster centres, number of clusters
    KM <- kmeans(gapminder.SoCL.2011.s,i,iter.max=2509,nstart=1)
    errs[j] <- sum(KM$withinss)
    DBI[j] <- Davies.Bouldin(KM$centers, KM$withinss, KM$size)
  }

  m.errs[i - 1] = mean(errs)
  s.errs[i - 1] = sd(errs)
  m.DBI[i - 1] = mean(DBI)
  s.DBI[i - 1] = sd(DBI)

  plot(pc.df.agg.data,col=KM$cluster, pch=KM$cluster, main=paste(i,"clusters"),
       xlab="PC1", ylab="PC2")
}
```



The average within sum of squares curve and the average Davies-Bouldin curves are also provided, with 95% confidence intervals.

```
# WSS
MSE.errs_up = m.errs + 1.96 * s.errs / sqrt(N)
MSE.errs_low = m.errs - 1.96 * s.errs / sqrt(N)

plot(2:(max.cluster), m.errs[1:(length(m.errs)-1)], main = "Within SS", xlab="", ylab="")
lines(2:(max.cluster), m.errs[1:(length(m.errs)-1)])
par(col = "red")

lines(2:(max.cluster), MSE.errs_up[1:(length(MSE.errs_up)-1)])
lines(2:(max.cluster), MSE.errs_low[1:(length(MSE.errs_low)-1)])

# DBI
MSE.DBI_up = m.DBI + 1.96 * s.DBI / sqrt(N)
MSE.DBI_low = m.DBI - 1.96 * s.DBI / sqrt(N)

par(col = "black")
plot(2:(max.cluster), m.DBI[1:(length(m.DBI)-1)], main = "Davies-Bouldin", xlab="", ylab="")
lines(2:(max.cluster), m.DBI[1:(length(m.DBI)-1)])
par(col="red")

lines(2:(max.cluster), MSE.DBI_up[1:(length(MSE.DBI_up)-1)])
lines(2:(max.cluster), MSE.DBI_low[1:(length(MSE.DBI_low)-1)])
```



Where is the Davies-Bouldin index maximized?

```
(i_choice <- which(m.DBI==max(m.DBI[1:(length(m.DBI)-1)]))+1)
```

```
[1] 9
```

The WSS curve does not yield much information, but the DBI curve suggests that both $k = 3$ and $k = 9$ could be good parameter choices. With parsimony considerations in mind, we might elect to use $k = 3$, but if the results are too simple or if signs of instability appear,[36] $k = 9$ might prove to be a better choice in the end.

36: Recall that $k-$means is a stochastic algorithm.

22.4 Advanced Clustering Methods

In the rest of this chapter, we present representative clustering algorithms from the remaining families.[37]

37: Substantially more information on the topic can be found in [2, 18, 48].

22.4.1 Density-Based Clustering

The assumptions of the $k-$means algorithm imply that the clusters that it finds are usually **Gaussian**.[38] But this is not always a desired outcome.

38: That is, blob-like.

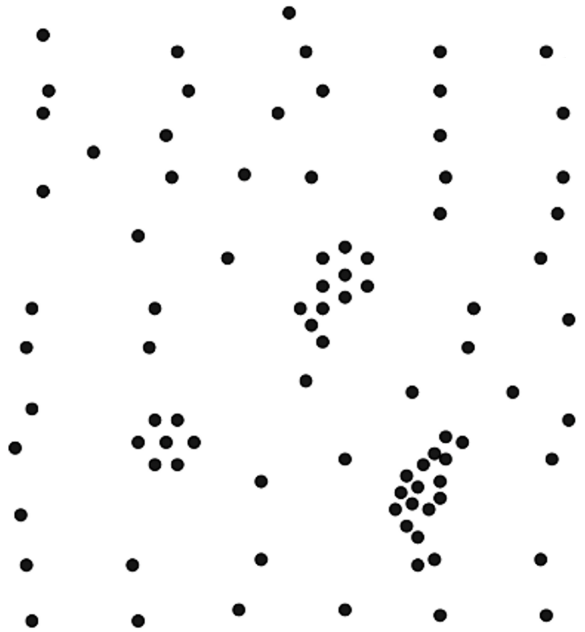
In **density-based clustering**, it is the **density** of observations and the **connectivity** of the accompanying clustering network that determine the number and location of clusters.[39] Popular density-based clustering algorithms include DBSCAN, DENCLUE, OPTICS, CHAMELEON, etc.

39: We will discuss these further in the next section.

Once density has been defined in a meaningful way, [40] density-based algorithms are straightforward to apply (see [2, 38, 34, 37, 17]).

40: Which depends on a number of contextual factors.

Density How do we measure density? Intuitively, we can recognize areas of **low density** and **high density** in the (artificial) dataset below.



As the saying goes, "birds of a feather flock together"; it should not come as a surprise that areas of higher density could be viewed as **clusters** in the data. In that context, if $\mathbf{\Psi} \subseteq \mathbb{R}^n$ is an $n-$dimensional **sub-manifold** of $\mathbb{R}^n$, we could define the **density** of $\mathbf{\Psi}$ around $\mathbf{x}$ by, say,

$$\text{density}_{\mathbf{\Psi}}(\mathbf{x}; d) = \lim_{\varepsilon \to 0^+} \frac{\text{Vol}_n(B_d(\mathbf{x}, \varepsilon) \cap \mathbf{\Psi})}{\text{Vol}_n(B_d(\mathbf{x}, \varepsilon))},$$

where

$$B_d(\mathbf{x}, \varepsilon) = \{\mathbf{y} \in \mathbb{R}^n \mid d(\mathbf{x}, \mathbf{y}) < \varepsilon\}$$

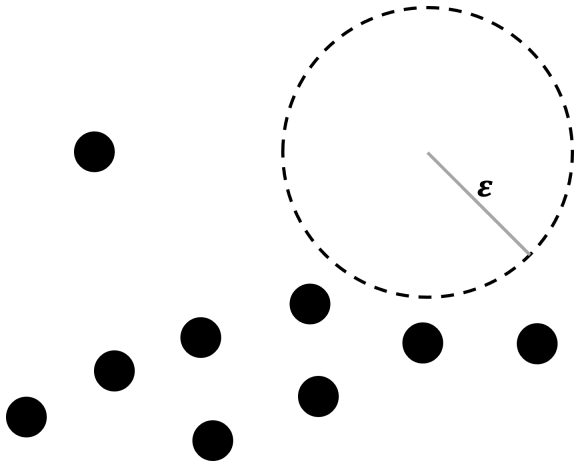
and

$$\text{Vol}_n(A) = n - \text{volume of } A \text{ in } \mathbf{R}^n.$$

DBSCAN In practice, the dataset $\mathbf{X}$ is usually a **discrete subset** of $\mathbb{R}^n$, and the limit definition above cannot apply. **Density-based spatial clustering of applications with noise** (DBSCAN) estimates the density at an observations $\mathbf{x} \in \mathbf{X}$ as follows: we pick a "reasonable" value of $\varepsilon^* > 0$ and set

$$\text{density}_{\mathbf{X}}(\mathbf{x}; d) = |B_d(\mathbf{x}, \varepsilon^*) \cap \mathbf{X}| \,.$$

The outcome depends, of course, on the choice of ε^* and the distance d.



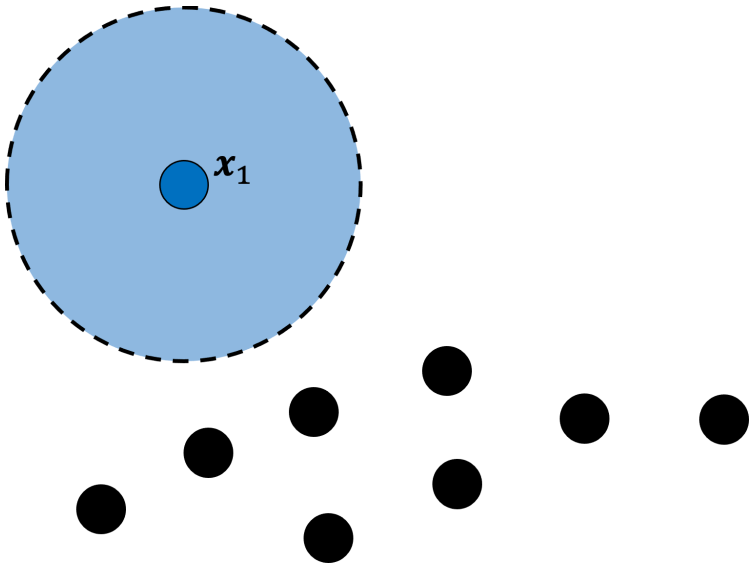
DBSCAN also requires a connectivity parameter: the **minimum number of points** *minPts* in

$$V_{\mathbf{x}} = B_d(\mathbf{x}, \varepsilon^*) \cap [\mathbf{X} \setminus \{\mathbf{x}\}]$$

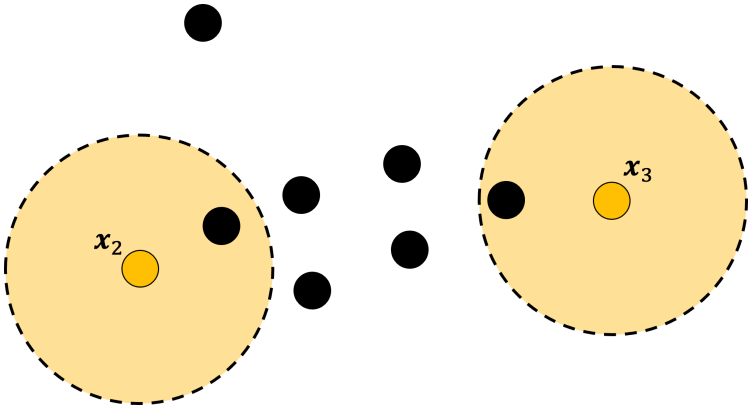
(excluding $\mathbf{x}$). If $|V_{\mathbf{x}}| \geq minPts$, the observations in $V_{\mathbf{x}}$ are said to be **within reach of** (or reachable from) $\mathbf{x}$.

In other words, for a given choice of d, ε^*, and *minPts*, there are three **types of observations** in $\mathbf{X}$:

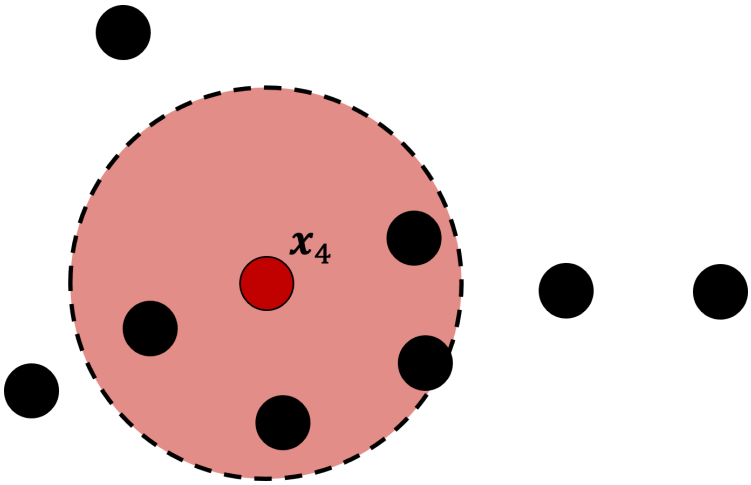
- **outliers** are observations that are not within reach of any of the other observations, such as $\mathbf{x}_1$ below:



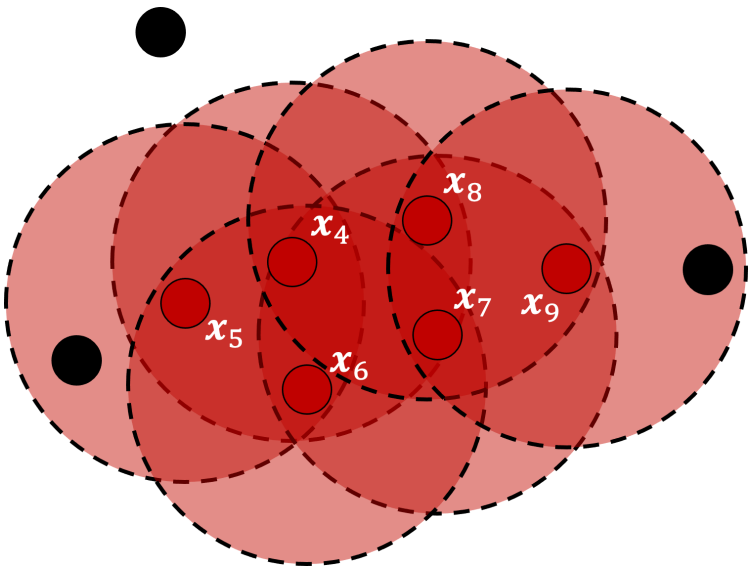
- **reachable** (non-core) **observations** are observations that are within reach of fewer than *minPts* other observations, such as $\mathbf{x}_2$ and $\mathbf{x}_3$ below (with *minPts* = 3):



- **core observations** are within reach of at least *minPts* other observations, such as $\mathbf{x}_4$ below (with *minPts* = 3):



There are other core points: $\mathbf{x}_5$, $\mathbf{x}_6$, $\mathbf{x}_7$, $\mathbf{x}_8$, and $\mathbf{x}_9$.



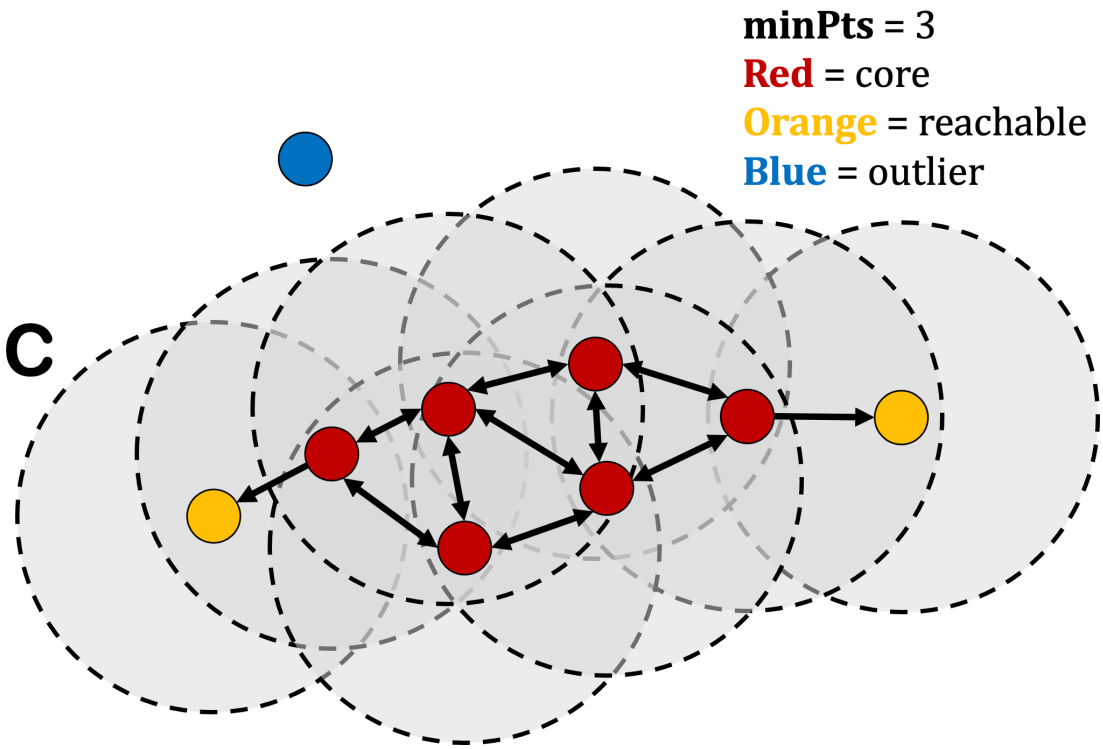


Figure 22.13: Density path connection in a DBSCAN cluster C.

Reachability is **not a symmetric relation**: no observation is reachable from a non-core point (a non-core point may be reachable, but nothing can be reached from it).

We can build a new symmetric relation on non-outlying observations on the basis of reachability, however:

$$\mathbf{p}, \mathbf{q} \in \mathbf{X} \setminus \{\text{outliers}(\mathbf{X})\}$$

are said to be **density-connected** for $\varepsilon^* > 0$ and d if there is an observation $\mathbf{o} \in \mathbf{X}$ such that $\mathbf{p}, \mathbf{q} \in V_{\mathbf{o}}$, with $|V_{\mathbf{o}}| \geq minPts$.

The same $\mathbf{p}, \mathbf{q}$ are said to be **density-connected in a path** if either they are density-connected or if there is a sequence of observations

$$\mathbf{p} = \mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{k-1}, \mathbf{r}_k = \mathbf{q}$$

such that $\mathbf{r}_{i-1}, \mathbf{r}_i$ is density-connected for all $i = 1, \ldots, k$.

That the latter is a relation on $\mathbf{X} \setminus \{\text{outliers}(\mathbf{X})\}$ is clear:

- it is **reflexive** as every $\mathbf{x} \in \mathbf{X} \setminus \{\text{outliers}(\mathbf{X})\}$ is either reachable or a core observation, so that $\exists \mathbf{o}_{\mathbf{x}} \in \mathbf{X}$ with $\mathbf{x} \in V_{\mathbf{o}_{\mathbf{x}}}$ and $|V_{\mathbf{o}_{\mathbf{x}}}| \geq$ minPts, and so $\mathbf{x}$ is density-connected to itself;
- it is **symmetric** and **transitive** by construction.

DBSCAN clusters are, essentially, composed of observations that are density-connected in a path.

In the image above, arrows represent density-connection: each orange observation is within reach of a red one, but no observation can be reached from the orange points.

Algorithm DBSCAN clusters are grown as follows:

1. select an observation **at random** that has yet to be assigned to a cluster, from the list of not previously selected observations;
2. determine the selected observation's type (outlier, non-core, core);
3. if the observation is an outlier or a non-core point, assign it to the **noise cluster**;
4. else, build its network of density-connected paths;
5. assign all observations in the network to a **stand-alone cluster**;
6. repeat steps 1 to 5 until all points have been assigned to a cluster.

All points within a cluster are **mutually density-connected in a path**. If a point is reachable from any point of the cluster, it is part of the cluster as well. An illustration of the DBSCAN algorithm is provided in Figure 22.14.

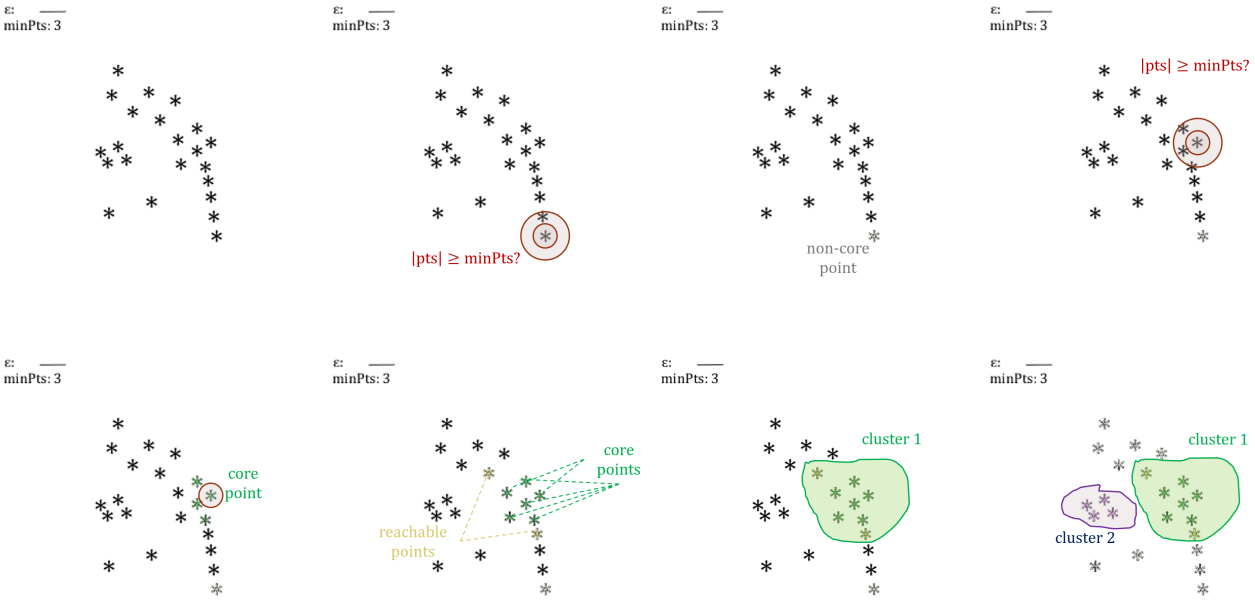


Figure 22.14: Illustration of DBSCAN on an artificial dataset (top, left). The parameters ε and *minPts* are shown in each display. We select a point at random (second image, top row); it is not a core point as its $\varepsilon-$neighbourhood does not contain more than *minPts* observations (excluding the selected point itself); it is assigned to the noise cluster. We select another point at random (top, right); that one is core point, as its $\varepsilon-$neighbourhood contains 4 observations. All its density-connected observations are shown in green (bottom, left). Its network of density-connected paths is shown in green, for the core observations, and in light green, for the reachable observations (bottom row, second image); they make up cluster 1 (bottom row, third image). Continuing on this way, we obtain 2 clusters and noisy observations (bottom, right).

The observations in the noise cluster are typically identified as **outliers**, making DBSCAN a reasonable unsupervised learning approach for anomaly detection (see Chapter 26).

Note that clusters, by definition, must contain **at least** one core point. Small groups of observations that are not density-connected to any core points will then also be assigned to the noise cluster. A non-core point that has been assigned to the noise cluster may end up being assigned to a stand-alone cluster at a later stage (but the opposite cannot occur).

It is possible for two clusters to **share** non-core points, in which case the points in question are randomly assigned (the random order of selection in step 1 may affect the results); consequently, some clusters may end up containing **fewer than** *minPts* observations.

Comments The main **advantages** of DBSCAN are that:

- there is no need to specify the **number** of clusters to find in the data;
- clusters of **arbitrary shapes** can be discovered;
- observations that are "noisy"/outlying are not forced into a cluster;
- the clusters are **robust** with respect to outliers, and
- it only requires two parameters ($\varepsilon^* > 0$ and $minPts$) to run properly, which can be set by domain experts if the data is well understood.

In general, it is suggested to use $minPts \geq p + 1$, with larger values being preferable for **noisy** datasets, or $minPts \geq 2p$ for large datasets or sets with duplicates. Meanwhile, the choice of $\varepsilon^* > 0$ should take into account that if it is too small, a large portion of the observations will be **assigned to the noise cluster**; but if it is too large, a majority of observations will be found in a **single cluster**. Small values are preferable, but how small is too small?

The parameter choices have a large impact on the DBSCAN results, as does the choice of the distance function, which should take place **before** ε^* is selected to avoid **data dredging** and "begging the question". Given that DBSCAN can handle globular clusters as well as non-globular clusters, why would we not always use it?

One important reason relates to **computational efficiency**. For a dataset $\mathbf{X}$ with n observations, the basic $k-$means algorithm has order $O(nk)$, whereas the most efficient versions of DBSCAN algorithm has order $O(n \log n)$. Thus, when n increases, the DBSCAN runtime increases faster than the $k-$means runtime.

Another reason is that DBSCAN works well when the density of clusters is assumed to be **constant**.

Most of us would agree that there are **two clusters** in the image above – a loose one in the bottom/left corner, and a tight one in the top/right corner – as well as some **outliers** around the tight cluster, but no combination of $\varepsilon^* > 0$ and $minPts$ can allow DBSCAN to discover this structure: either it finds no outliers, or it only finds the one tight cluster.

Example We re-visit the (scaled) 2011 Gapminder dataset: we use Euclidean dissimilarity in this example, but the `dbscan()` function from the `fpc` package in `R` can accommodate other metrics: we first compute the corresponding distance matrix and specify `method="dist"` instead of `method="raw"` in the function call.

We will use 9 combinations of parameters

$$(\varepsilon^* \in \{0.75, 1, 1.25\}) \times (minPts \in \{6, 10, 15\}).$$

```
set.seed(0) # for replicability
dbscan1 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 0.75, MinPts = 6)
dbscan2 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 1.0, MinPts = 6)
dbscan3 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 1.25, MinPts = 6)
dbscan4 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 0.75, MinPts = 10)
dbscan5 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 1.0, MinPts = 10)
dbscan6 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 1.25, MinPts = 10)
dbscan7 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 0.75, MinPts = 15)
dbscan8 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 1.0, MinPts = 15)
dbscan9 <- fpc::dbscan(gapminder.SoCL.2011.s, eps = 1.25, MinPts = 15)
```

No doubt there are more efficient ways to go through the 9 combinations, but this will do for the purpose of illustration.

```
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan1$cluster)),
                diag=list(continuous=my_dens))
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan2$cluster)),
                diag=list(continuous=my_dens))
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan3$cluster)),
                diag=list(continuous=my_dens))
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan4$cluster)),
                diag=list(continuous=my_dens))
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan5$cluster)),
                diag=list(continuous=my_dens))
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan6$cluster)),
                diag=list(continuous=my_dens))
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan7$cluster)),
                diag=list(continuous=my_dens))
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan8$cluster)),
                diag=list(continuous=my_dens))
GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(dbscan9$cluster)),
                diag=list(continuous=my_dens))
```

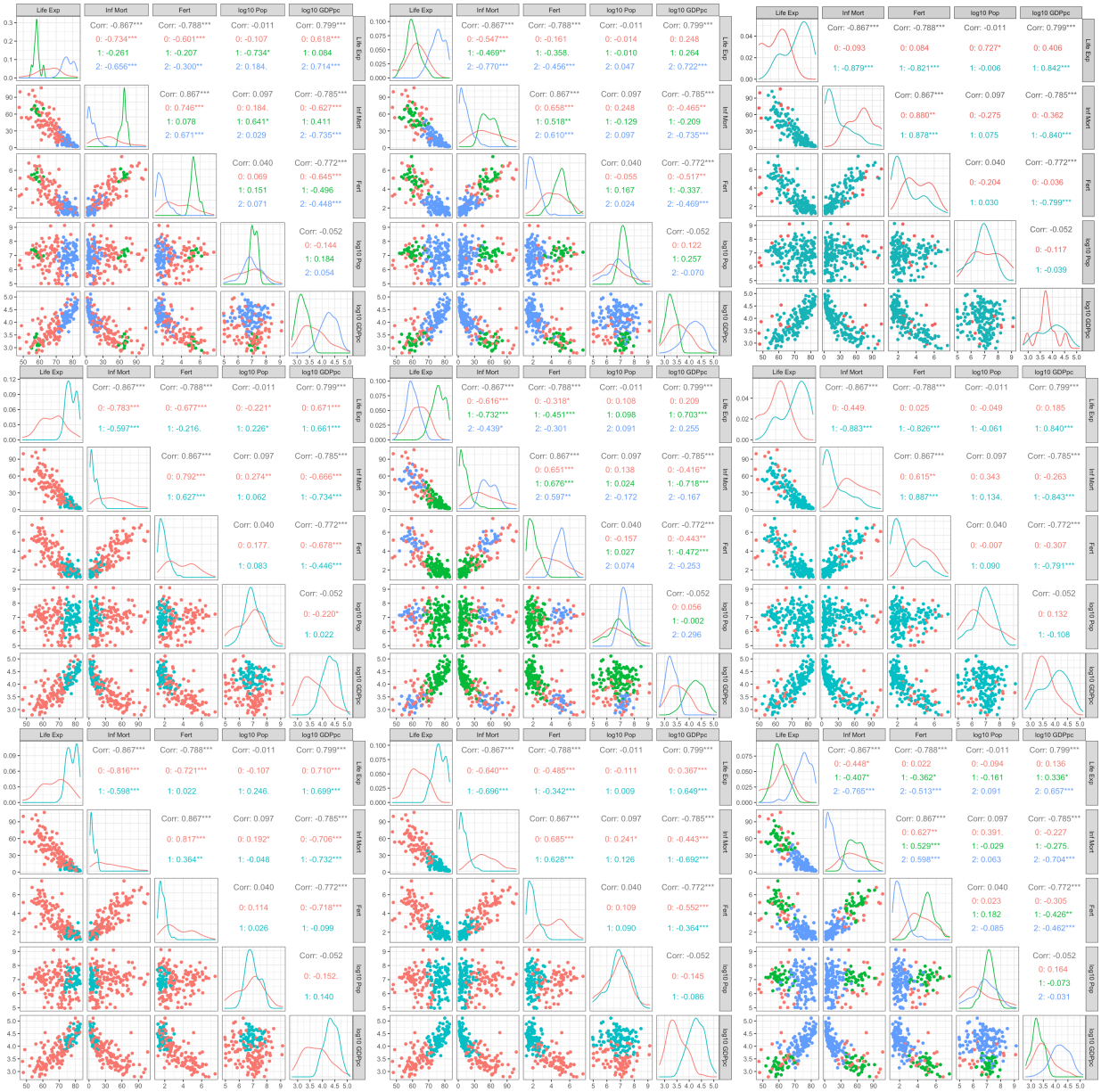


Figure 22.15: Realizations of DBSCAN on the (scaled) 2011 Gampinder data: $\varepsilon = 0.75$ (first column), 1 (second column), 1.25 (third column); *minPts* = 6 (first row), 10 (second row), 15 (third row).

The noisy observations are shown in red: one immediate insight is that the number of outlying observations **decreases** as ε^* increases, which is as expected. Another insight is that the number of noisy observations **increases** as *minPts* increases, which is again not surprising.

If we compare the shape of the DBSCAN clusters with those of the $k-$means and HC clusters, we notice that the option of identifying observations as noisy – coupled with the "right" combination of parameters – creates "reasonable" clusters, that is to say, clusters for which we do not have to stretch our ideas about what clusters ought to look like: the problematic observations[41] are simply explained away as outliers.

41: Like China and India in regards to population, say.

The various runs find either 1 or 2 stand-alone clusters (as well as noisy observations), but that can change if we use different parameter values.

We can also determine if the cluster observations are core or non-core observations. In the realization with $\varepsilon^* = 1$ and $minPts = 6$, we have:

	noise	**cluster 1**	**cluster 2**
outlier	34	–	–
reachable	–	10	17
core	–	20	103
total	**34**	**30**	**120**

22.4.2 Spectral Clustering

At a fairly coarse level, clustering algorithms are divided along those focusing on **compactness** and those focusing on **connectivity**.

Compactness methods are usually variants of k **Nearest Neighbours** (kNN) methods (see Section 21.1.3), and are effective when there are distinct clumps in the data. We can make specific assumptions about the distribution of the different clusters ahead of time (as in the next section), but compact methods struggle to achieve meaningful results in scenarios where groups are not **linearly separable**.

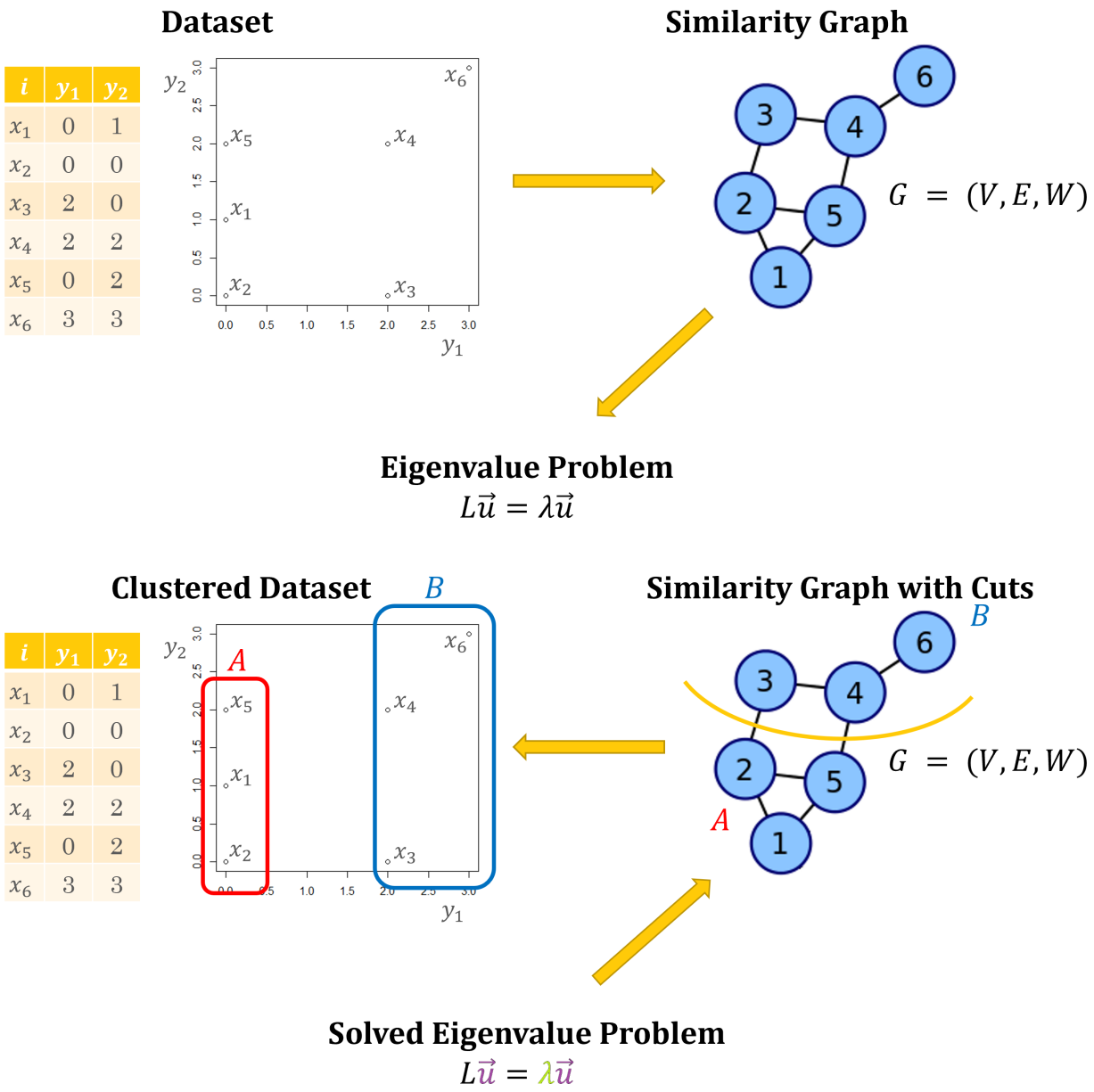
In cases where we have little to no knowledge of the dataset, making assumptions about the distributions of clusters can lead to invalid clustering schemes; in such cases, connectivity-based methods have been shown to work reasonably well [30, 18].

Connectivity methods, such as DBSCAN, focus on dividing observations into groups based on their **similarity graphs**; observations that are quite different in their features[42] may end up in the same cluster if there is a chain of sufficiently similar observations linking them.

42: And as such would be differentiated using **compactness** methods.

Connectivity methods require fewer initial assumptions, but their use can be harder to justify mathematically. The validity of such methods can only be determined *post hoc*.

Spectral clustering is a connectivity method that has become quite popular in practice; in a nutshell, we transform the dataset into its **similarity graph** and convert the latter into an **eigenvalue problem**. We then solve the eigenvalue problem, convert the solution into a **graph cut**, and then translate the cut back into dataset **clusters** (as illustrated in Figure 22.16).



i	y_1	y_2
x_1	0	1
x_2	0	0
x_3	2	0
x_4	2	2
x_5	0	2
x_6	3	3

Figure 22.16: Schematics of spectral clustering. We extract the similarity graph of a dataset, which gives rise to an eigenvalue problem (top). The eigenenvalue problem is then solved, which suggests an 'optimal' graph cut, which in turns leads to data clusters (bottom).

Before we start delving into the spectral clustering algorithm, we must discuss a few concepts relating to graphs and linear algebra.[43]

43: These concepts are covered in just enough depth to provide an intuition about the algorithm.

Graphs and Cuts A **graph** is an object which connects **nodes** (or **vertices**) together through **edges**. The edges have **weights** and can also be **directed**. In certain cases, we may assume that all edge weights are identical and bidirectional, which is equivalent to saying that the edges just represent that a relationship exists.

Airports (vertices) and flight paths (edges) form a graph in transportation networks, as do people (vertices) and relationships (edges) in social networks; the edges can be weighted according to flight frequency and/or directed according to their origin and destination, say, in the transportation example.

In the social network example, they could be weighted according to frequency of communication and/or directed according to who follows who on some app.

The link with clustering is that once a similarity measure w has been selected, a dataset can be represented by a **similarity graph** $G = (V, E, W)$:

1. observations $\mathbf{x}$ correspond to **vertices** $v \in V$;
2. if $i \neq j$, vertices $v_i, v_j \in V$ are connected by an **edge** $e_{i,j} = 1$ if the **similarity weight** $w_{i,j} = w(\mathbf{x}_i, \mathbf{x}_j) > \tau$ for a predetermined **threshold** $\tau \in [0, 1)$, and by no edge ($e_{i,j} = 0$) otherwise;[44]
3. the edges $(e_{i,j})$ form the **adjacency matrix** E;
4. the weights $(w_{i,j})$ form the **similarity matrix** W;
5. the **(diagonal) degree matrix** D provides information about the number of edges attached to a vertex: $d_{i,i} = \sum_{j=1}^{n} e_{i,j}$.

44: Note that, by convention, $w_{i,i} = 0$ for all i.

As an example, we could use the **Gower similarity measure**

$$w(\mathbf{x}_i, \mathbf{x}_j) = 1 - \frac{1}{p} \sum_{k=1}^{p} \frac{|x_{i,k} - x_{j,k}|}{\text{range of } k\text{th feature in } \mathbf{X}}$$

on the dataset found in Figure 22.16; the ranges of X_1 and X_2 are both $r_1 = r_2 = 3$, so that

$$\begin{aligned} w_{3,4} = w_{4,3} = w(\mathbf{x}_3, \mathbf{x}_4) &= 1 - \frac{1}{2}\left(\frac{|x_{3,1} - x_{4,1}|}{r_1} + \frac{|x_{3,2} - x_{4,2}|}{r_2}\right) \\ &= 1 - \frac{1}{2}\left(\frac{|2-2|}{3} + \frac{|0-2|}{3}\right) = 1 - \frac{1}{2} \cdot \frac{2}{3} = \frac{2}{3}; \end{aligned}$$

the similarity matrix as a whole is

$$W = \begin{pmatrix} 0 & 5/6 & 1/2 & 1/2 & 5/6 & 1/6 \\ 5/6 & 0 & 2/3 & 1/3 & 2/3 & 0 \\ 1/2 & 2/3 & 0 & 2/3 & 1/3 & 1/3 \\ 1/2 & 1/3 & 2/3 & 0 & 2/3 & 2/3 \\ 5/6 & 2/3 & 1/3 & 2/3 & 0 & 1/3 \\ 1/6 & 0 & 1/3 & 2/3 & 1/3 & 0 \end{pmatrix}.$$

If we use a threshold value of $\tau = 0.6$, say, then the adjacency matrix is

$$E = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix},$$

and the degree matrix is

$$D = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

The degree matrix can also be read directly from the similarity graph (which depends on the threshold τ), by counting the number of edges at each node (see Figure 22.16).

A **graph cut** is the process by which we remove edges from the graph and separate the vertices into into groups (or **sub-graphs**).

The clustering task is to separate the nodes into multiple groups by **minimizing the total weight of the edges** we have to break in the process (i.e., making sure that the groups are as dissimilar as possible). This is also known as the **minimum cut problem** (MinCut).[45]

45: This cannot be the entire story, however, as we can minimize the total weight of broken edges by simply ... not cutting any edges. Indeed, there are other approaches: **Normalized Cut** (actually used in practice), Ratio Cut, Min-Max Cut, etc.

This task is NP-Hard, which means that there is no theoretically guaranteed efficient way to do so, in comparison to simply testing every possible cut and finding the minimum weight. This is problematic: for datasets with n observations, the number of cuts is **bounded below** by 2^n (when we only consider 2−cuts); when n is relatively small, the overall number of cuts to consider remains manageable, but for nearly all reasonable datasets, the size of n turns this task into an exercise in futility.

The clustering approach generalizes the MinCut problem (or any of the other problems) by imposing some properties on the similarity graph to ensure that we can approximate the true MinCut solution in a **computationally efficient** manner.[46]

46: The spectral MinCut solution is not guaranteed to be the true MinCut solution, but it usually is close enough to be an acceptable approximation.

Formally, the MinCut problem involves finding a **partition** $\{A_1, ..., A_k\}$ of G which minimizes the objective function

$$\text{Cut}(A_1, ..., A_k) = \frac{1}{2} \sum_{i=1}^{k} \mathcal{W}(A_i, \overline{A_i}),$$

where

$$\mathcal{W}(A, B) = \sum_{i \in A, j \in B} w_{i,j}$$

and $\overline{A}$ is the (set-theoretic) complement of A. The factor $\frac{1}{2}$ is there to remove double-counted edges.

The spectral clustering approach instead solves the **Normalized Cut** (NCut) problem, which is similar to the MinCut problem except that we

are minimizing the weight of edges escaping a cluster relative to the total weights in the cluster.[47]

47: For more information about this abstraction, which actually links a variant of Kernel PCA to spectral clustering, consult [7].

In the NCut problem, the **objective function** is

$$J_{\text{NCut}}(A, B) = \text{Cut}(A, B)\left(\frac{1}{\text{Vol}(A)} + \frac{1}{\text{Vol}(B)}\right),$$

where

$$\text{Vol}(C) = \sum_{i \in C} w_{i,*};$$

in a first pass, we seek to **minimize** J_{NCut} against the set of **all possible partitions** (A, B) of G. The procedure can be repeated as often as necessary on the cluster sub-graphs.

Intuitively, J_{NCut} is small when the observations **within** each sub-graph are **similar** to one another ($\text{Vol}(A)$, $\text{Vol}(B)$ are large) and the observations **across** are dissimilar to one another ($\text{Cut}(A, B)$ is small).

On the plus side, takes into consideration the **size of the partitioned groups** and **intra-group variance**, and tends to avoid isolating vertices, but it is not any easier to solve than the MinCut problem. So why do we even bring it up in the first place? As it happens, we can provide an approximation to the NCut solution using **purely algebraic means**.

Similarity, Degree, and Laplacian Matrices There are different ways to construct a **graph** representing the relationships between the dataset's observations. We can use:

- **fully connected** graphs, where all vertices having non-zero similarities are connected to each other;
- $r-$**neighbourhood** graphs, where each vertex is only connected to the vertices falling inside a ball of radius r (according to some distance metric d), where r has to be tuned to capture the local structure of data;
- k **nearest neighbours** graphs (and variants), where each vertex is connected to its k nearest neighbours (again, according to some distance metric d), with k pre-selected, and
- **mixtures of** $r-$**neighbourhood and** k**NN** graphs, to better capture sparsity in the data.

The similarity measure w is usually picked from a list that includes: Gaussian (most common), cosine, fuzzy, Euclidean, Gower, etc. The similarity matrix W is symmetric and has zeros along the diagonal; its non-diagonal entries represent the **similarity strength** between the corresponding graph vertices.[48]

48: And so beteween the corresponding observations in the dataset.

We have discussed previously how to build the adjacency matrix E from W and a threshold $\tau \in [0, 1)$. The only component of a graph that similarity matrices do not directly capture are the **degrees** of each vertex, the number of edges that enter it.[49]

49: We are viewing the similarity graph as **undirected**.

The **diagonal** of the degree matrix D holds that information for each vertex. We can combine W and D (or E and D) to create a matrix L known as the **Laplacian**, which has properties linked to the topology of the similarity graph.

The **Laplacian** of a graph is defined by

$$L_0 = D - \Theta, \quad \Theta \in \{E, W\};$$

the **symmetric Laplacian** by

$$L_S = D^{-1/2} L D^{-1/2} = \mathbf{I}_n - D^{-1/2} \Theta D^{-1/2},$$

and the **asymmetric Laplacian** by

$$L_A = D^{-1} L = \mathbf{I}_n - D^{-1} \Theta.$$

In all cases, the **off-diagonal entries** are non-positive, and the **diagonal entries** contain the degree of each node.

The Laplacians have the following useful properties:

- L_0, L_S are symmetric; L_A is not necessarily so;[50]
- all their eigenvalues are real and non-negative;
- every row and column adds up to 0, which means that $\lambda_0 = 0$ is the smallest eigenvalue of each Laplacian (hence they are singular and cannot be inverted);
- the number of connected components in the graph is the **dimension of the nullspace** of the Laplacian associated to $\lambda_0 = 0$ (which may provide a first approximation to the number of clusters in $\mathbf{X}$), and
- the second smallest eigenvalue gives the graph's **sparsest cut**.[51]

50: Since the product of symmetric matrices is not necessarily symmetric.

51: This is not the same as the minimum cut which represents the cut that minimizes the number of edges separating two vertices, but instead represents the minimum ratio of edges across the cut divided by the number of vertices in the smaller half of the partition.

52: This notion will also play a role in Section 23.4.3.

Algorithm In the case of two clusters, the objective function J_{NCut} is minimized when finding the eigenvector $\mathbf{f}$ corresponding to the smallest **positive** eigenvalue of L, also known as the **spectral gap**.[52]

The clustering in the original data is recovered by sending $\mathbf{x}_i$ to A when $f_i > 0$ and $\mathbf{x}_j$ to B otherwise. This deterministic algorithm is a special case of the **spectral clustering algorithm** [28].

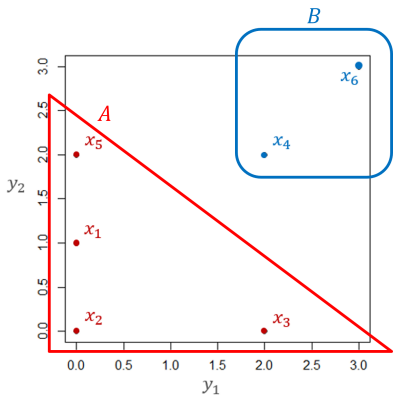
To split $\mathbf{X}$ into k clusters, we follow the steps below:

1. form a similarity matrix W and a degree matrix D using a threshold $\tau \in [0, 1)$;
2. construct a Laplacian L_ξ, $\xi \in \{0, S, A\}$, using $\Theta = W$;
3. compute the first k eigenvectors $\{\mathbf{u}_1, ..., \mathbf{u}_k\}$ of L_ξ corresponding to the k **smallest positive** eigenvalues of L_ξ;
4. construct the $n \times k$ matrix $\mathbf{U}$ containing the vectors $\{\mathbf{u}_1, ..., \mathbf{u}_k\}$ as **columns**;
5. normalize the rows of $\mathbf{U}$ into a matrix $\mathbf{Y}$ with rows $\{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ having unit length;
6. cluster the rows of $\mathbf{Y}$ into k clusters;
7. assign $\mathbf{x}_i$ to cluster j of $\mathbf{X}$ if $\mathbf{y}_i$ was assigned to cluster j in the preceding step.

Spectral clusters for the dataset of Figure 22.16, computed using the Laplacian and symmetric Laplacian, are shown in Figure 22.17.

From an experimental perspective, spectral clustering provides an attractive approach because it is easy to implement and reasonably fast, especially for sparse datasets: it is a **graph partitioning problem** that makes no initial assumptions on the form of the data clusters.

i	f_i	A/B
x_1	-0.39	A
x_2	-0.30	A
x_3	-0.18	A
x_4	0.03	B
x_5	-0.16	A
x_6	0.84	B



i	f_i	A/B
x_1	0.36	B
x_2	0.54	B
x_3	-0.03	A
x_4	-0.43	A
x_5	0.14	B
x_6	-0.61	A

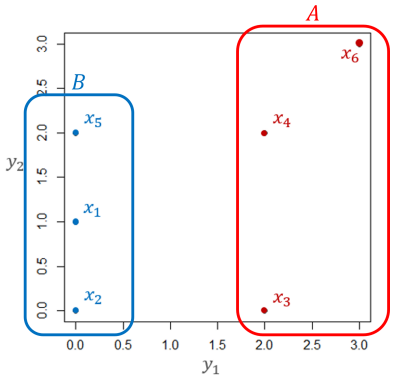


Figure 22.17: Two clusters for the artificial dataset: simple Laplacian (left); symmetric Laplacian (right).

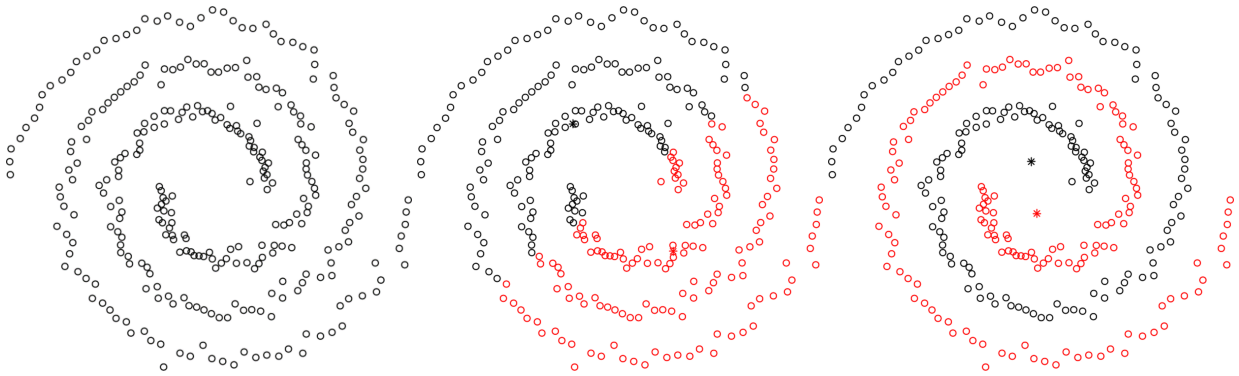


Figure 22.18: Comparing 2−means (middle) and spectral clustering with $k = 2$ (right) on the spirals dataset (left).

Spectral clustering has variants, which depend on the many choices that can be made at various points in the process:

1. **pre-processing** (choice of: number of cluster k, similarity measure w, threshold τ);
2. **spectral representation** (choice of Laplacian);
3. **clustering algorithm** (choice of compact-based, potentially non-deterministic algorithm to unleash on the rows of the representation $\mathbf{Y}$).

The **NJW algorithm** uses L_S for the spectral representation and k−means as a clustering approach. It can be interpreted as **kernalized** k−**means**: if we select a kernel which transforms the points to their mapped value in the Laplacian of the graph, then we (almost directly) recover spectral clustering [7].[53]

In Figure 22.18, the different outcomes of k−means and NJW are illustrated on the `spirals` dataset (available in `R`).

Practical Details and Limitations The most obvious practical detail in the implementation of spectral clustering is related to the construction of the similarity graph. In general, there is virtually no theoretical justification for determining what type of clustering approach to use; even after an approach has been selected, it can be quite difficult to choose appropriate parameter values.

53: DBSCAN can also fit within that framework, by picking a similarity method based on the radius that allows the graph to separate into different components. Then the multiplicity of $\lambda_0 = 0$ in the Laplacian gives the number of graph components, and these can be further clustered, as above.

In spectral clustering, there are considerations in favour of using **sparse similarity/adjacency matrix**: we seek to strike a balance between a Laplacian which is too densely connected, and one for which almost all of the observations are seen as dissimilar to one another. Another issue relates to the computational challenge of **finding the eigenvalues** of the Laplacian.

This can be done relatively efficiently if the matrix is sparse enough, however, which suggests using a relatively-high threshold τ; there are methods which help spectral clustering automatically tune for the best parameter values (including τ), but they take up a significant amount of resources [28].

Spectral clustering methods are extremely effective because they do not require assumptions about **distributions** and **centres**, are fairly **easy to implement**, and are **transparent** and **interpretable**.

However, they suffer from some of the same drawbacks as other clustering methods, namely when it comes to:

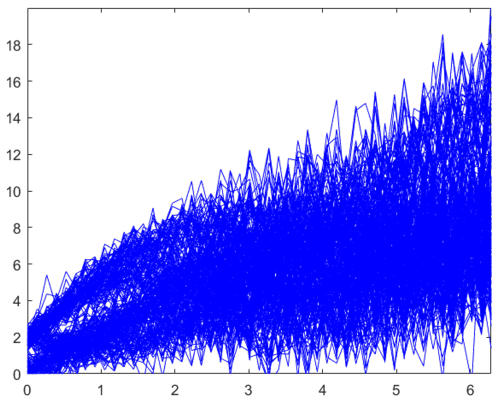
- selecting **initial parameter values**,
- run-times that **do not scale** with larger datasets, and
- determining optimal ways to visualize the results.

As in all clustering scenarios, analysts are faced with decisions at various levels of the process; they must be prepared to run multiple algorithms, in multiple configurations, in order to get a sense for the data structure.[54]

54: Some strategies specific to spectral clustering are presented in [28].

Examples In a first example, spectral clustering is used to segment greyscale images into different segments based on contrasting colours [43]. Figure 22.19 shows instances with high contrast, with fairly decent segmentation performance using NCut, Self-Tuning SC [52], and a proposed SC algorithm [43]; Figure 22.20 shows other instances with less contrast (resulting in a poorer segmentation with the same methods); Figure 22.21 shows the comparison in segmentations using the proposed SC algorithm when the same image is presented at different resolutions.

In the second example, consider a dataset of $n = 250$ times series, with $N = 60$ entries each (see below).



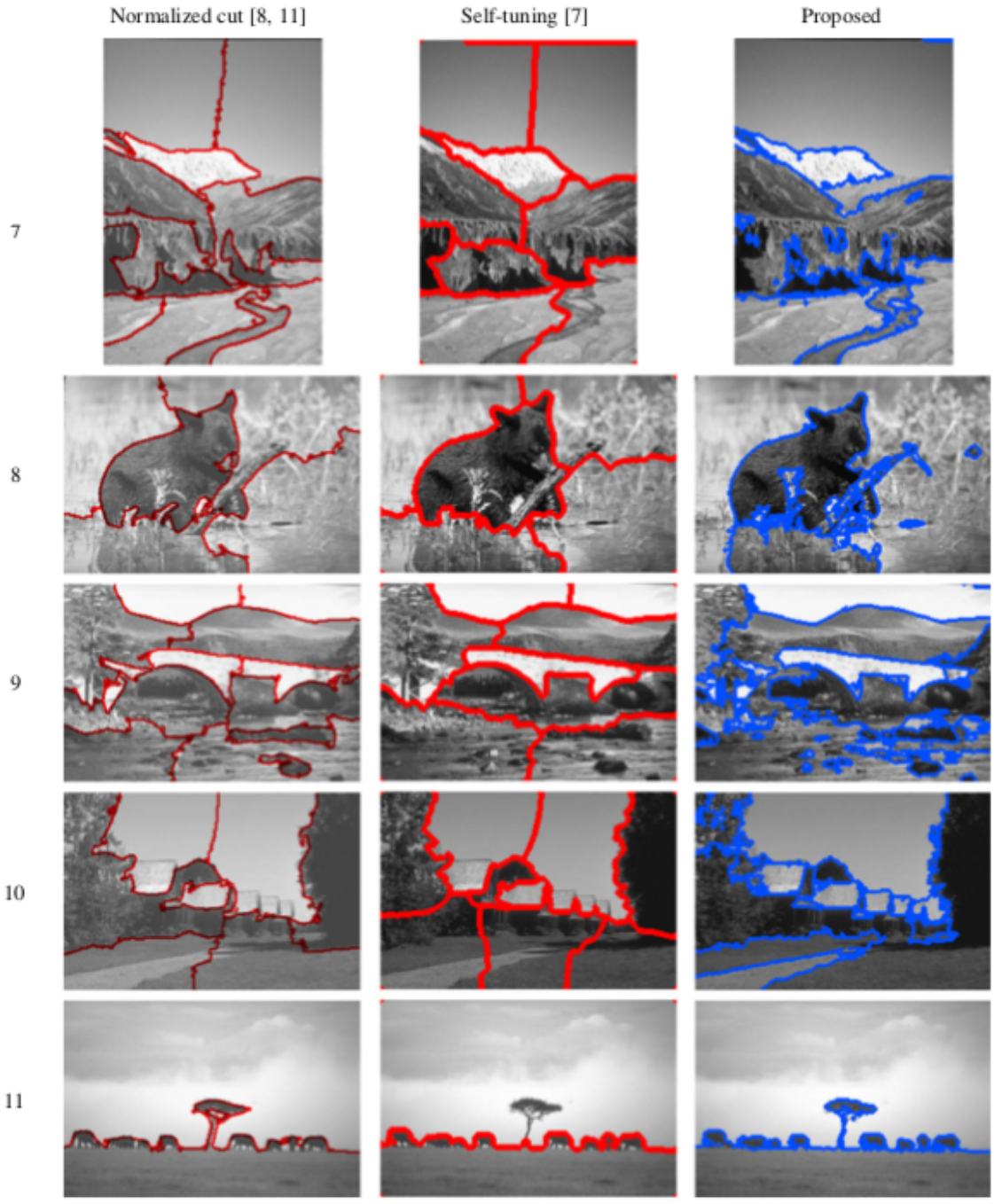


Figure 22.19: High-contrast image segmentation with spectral clustering [43].

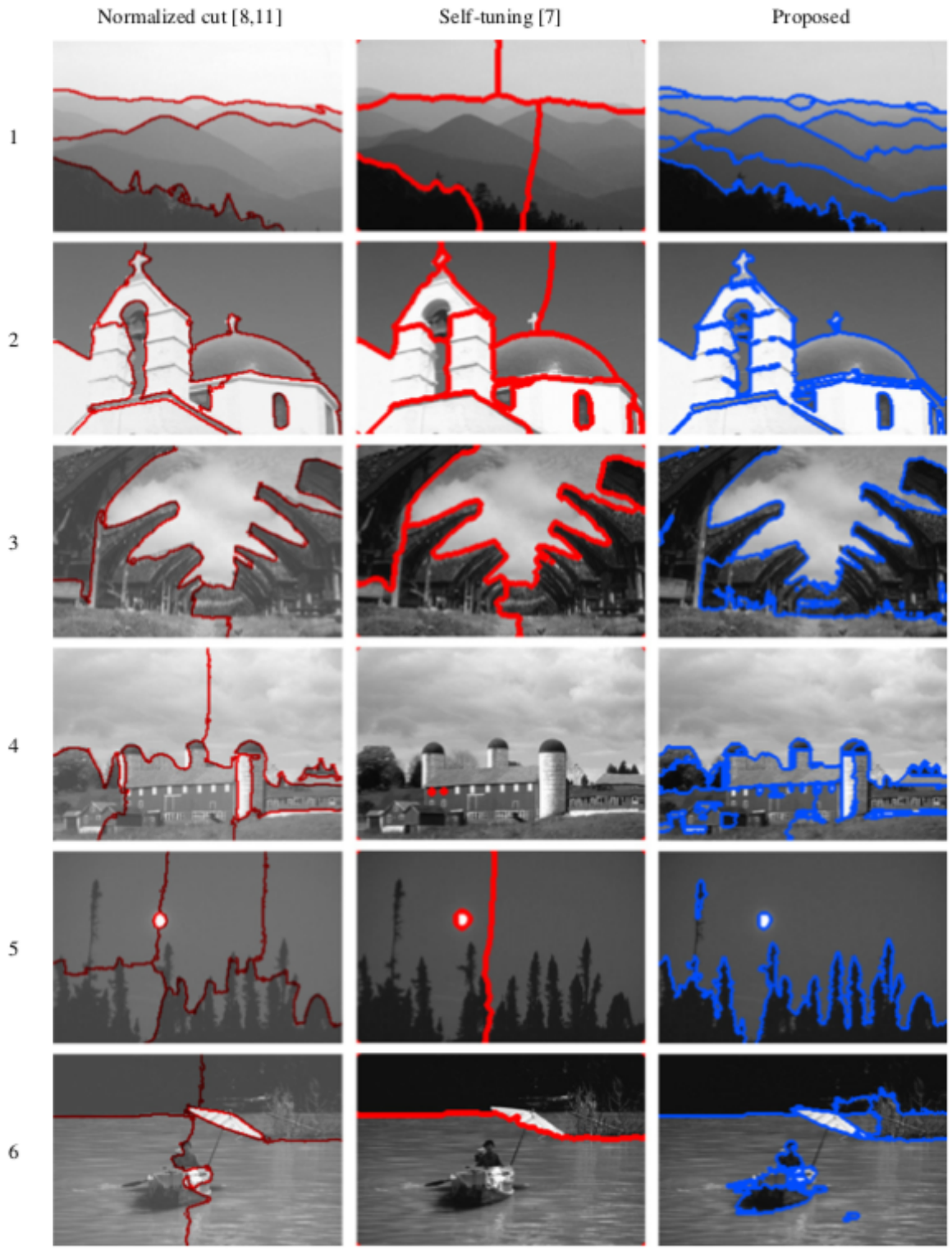


Figure 22.20: Low-contrast image segmentation with spectral clustering [43].

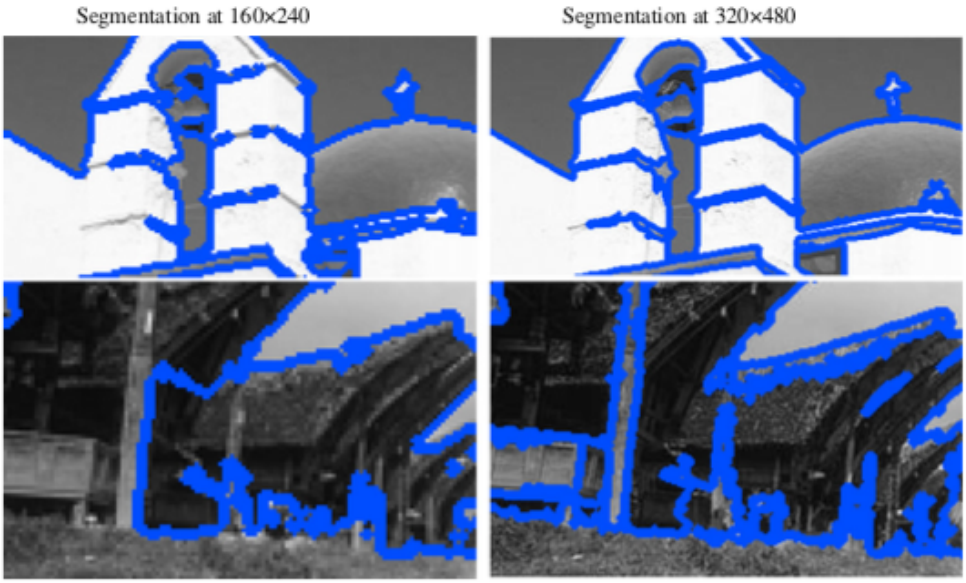


Figure 22.21: Spectral clustering image segmentation of images at different resolutions [43].

We use the **average absolute gap** as distance d:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{60} \sum_{\ell=1}^{60} |x_{i,\ell} - x_{j,\ell}|.$$

We build the **Gaussian similarity** measure

$$w(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{d^2(\mathbf{x}_i, \mathbf{x}_j)}{2\sigma^2}\right),$$

and we use the following parameter values

$$\sigma^2 = 300, \quad \tau = 0.9, \quad k = 5.$$

The spectral clustering results are quite appealing, as can be seen in the first realization of the NJW algorithm with $k = 5$ clusters. Note however that not every run of the algorithm yields an outcome that we would consider meaningful (see Figure 22.22).

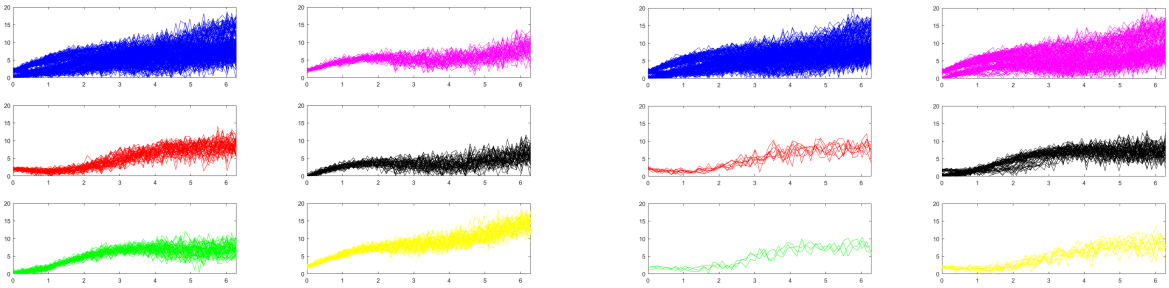
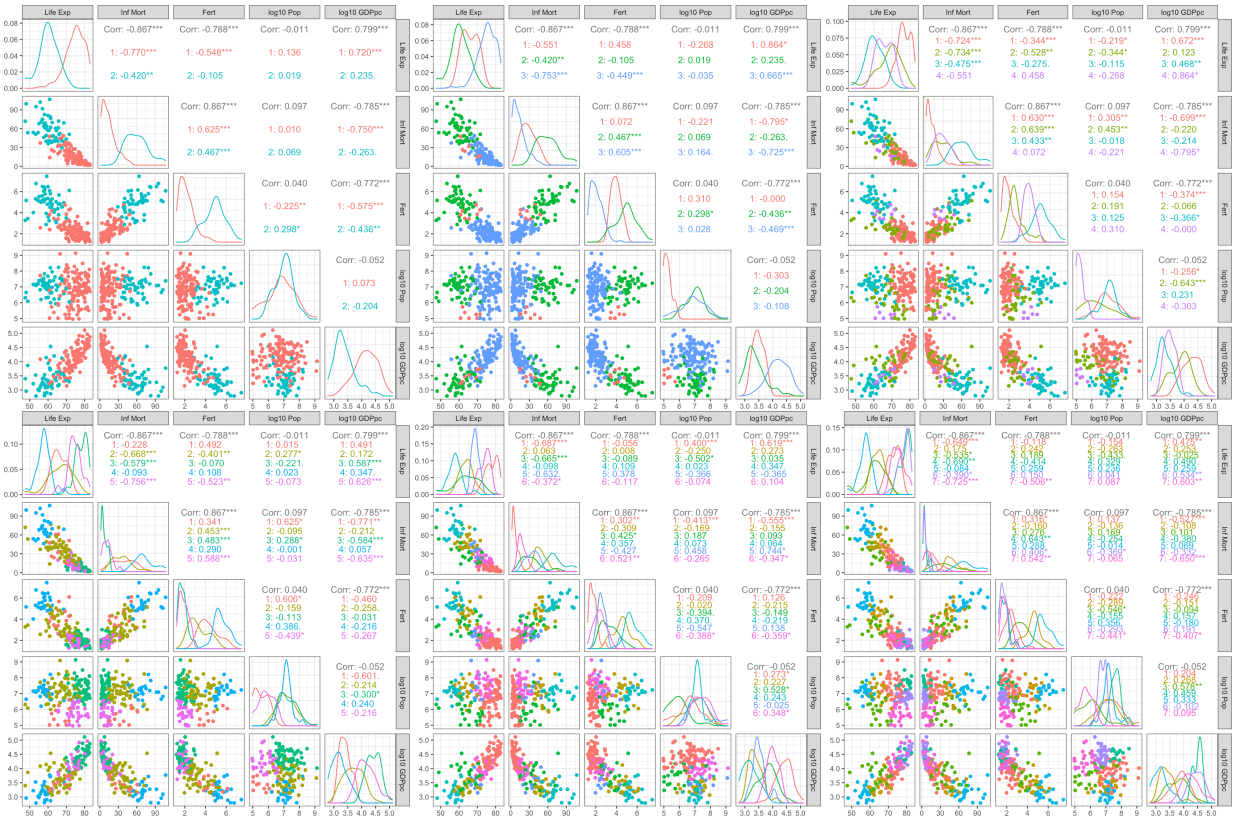


Figure 22.22: Two realizations of spectral clustering, using the NJW algorithm with $k = 5$; the original dataset is shown in blue. We see that the NJW algorithm has captured 5 clusters with different times series characteristics, which is an encouraging result (two leftmost columns); the $k-$means portion of the algorithm leads to different clusters, which appear to be of lower quality (two rightmost columns).

In the final example, we once again revisit the (scaled) 2011 Gapminder dataset using Euclidean dissimilarity. We use the `kernlab` implementation of the NJW algorithm found in `specc()`, with the default settings. We run one instance of the algorithm for $k = 2$ to $k = 7$ clusters.[55]

55: Assume that the libraries `ggplot2` and `GGally` hgave already been loaded.

```
sc.gapminder.2 <- kernlab::specc(as.matrix(gapminder.SoCL.2011.s), 2)
ggpairs(gapminder.SoCL.2011[,c(3:7)],
        aes(color=as.factor(sc.gapminder.2)), diag=list(continuous=my_dens))

sc.gapminder.3 <- kernlab::specc(as.matrix(gapminder.SoCL.2011.s), 3)
ggpairs(gapminder.SoCL.2011[,c(3:7)],
        aes(color=as.factor(sc.gapminder.3)), diag=list(continuous=my_dens))

sc.gapminder.4 <- kernlab::specc(as.matrix(gapminder.SoCL.2011.s), 4)
ggpairs(gapminder.SoCL.2011[,c(3:7)],
        aes(color=as.factor(sc.gapminder.4)), diag=list(continuous=my_dens))

sc.gapminder.5 <- kernlab::specc(as.matrix(gapminder.SoCL.2011.s), 5)
ggpairs(gapminder.SoCL.2011[,c(3:7)],
        aes(color=as.factor(sc.gapminder.5)), diag=list(continuous=my_dens))

sc.gapminder.6 <- kernlab::specc(as.matrix(gapminder.SoCL.2011.s), 6)
ggpairs(gapminder.SoCL.2011[,c(3:7)],
        aes(color=as.factor(sc.gapminder.6)), diag=list(continuous=my_dens))

sc.gapminder.7 <- kernlab::specc(as.matrix(gapminder.SoCL.2011.s), 7)
ggpairs(gapminder.SoCL.2011[,c(3:7)],
        aes(color=as.factor(sc.gapminder.7)), diag=list(continuous=my_dens))
```



None of our clustering attempts have found what one might call **natural groups** in the 2011 Gapminder data. We might not have hit on the right method yet... but at what point do we decide that the task is futile and no such groups exist in the first place?

22.4.3 Probability-Based Clustering

In contrast with the model-free approach of density-based clustering and spectral clustering, **probabilistic-based clustering** attempts to optimize the fit between the observed data and some **mathematical model** of clustering, with the assumption that the data is generated *via* a number of underlying probability distributions.

In practice, we assume that clusters are represented by **parametric probability distributions**, and the objective is to **learn the parameters** for each of these distributions. This assumption allows us to use probability theory to derive learning formulas for the parameters.[56]

56: We borrow extensively from Deng and Han's *Probabilistic Models for Clustering* chapter in [2].

Mixture Models The main underlying assumption of **mixture models** is that each observation is drawn (or generated) from one of several mechanisms (or components). In **model-based clustering**, we learn the parameters that provide the optimal fit to the data; in other words, we make a series of predictions about which component(s) generated each of the observations.

This naturally leads to **clusters**, all observations generated by a given component belonging to the same cluster. Formally, we let

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \in M_{n,p}(\mathbb{R}).$$

Assume that there are k mechanisms that generate data, and that each of them is determined by a vector of parameters $\boldsymbol{\theta}_\ell$, $1 \leq \ell \leq k$.

For $1 \leq j \leq n$, denote the probability of $\mathbf{x}_j$ being **generated by the ℓ−th mechanism**, $1 \leq \ell \leq k$, by

$$P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell).$$

The mixture vector $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_k)$ is a vector such that $\pi_\ell \in [0, 1]$ for all $1 \leq \ell \leq k$ and $\pi_1 + \cdots + \pi_k = 1$.

If $P(z_j = \ell) = \pi_\ell$, for $1 \leq \ell \leq k$, and if

$$P(\mathbf{x}_j \mid z_j = \ell) = P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell) \quad \forall j, \ell,$$

then the probability of observing $\mathbf{x}_j$ is

$$P(\mathbf{x}_j) = \sum_{\ell=1}^{k} \pi_\ell P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell) = \sum_{\ell=1}^{k} P(z_j = \ell) P(\mathbf{x}_j \mid z_j = \ell),$$

according to the **Law of Total Probability**.

In this set-up, we interpret z_j as the **cluster label** for $\mathbf{x}_j$. Alternatively, we could use

$$\mathbf{z}_j \in \{0, 1\}^k, \quad \|\mathbf{z}_j\|_2 = 1$$

to denote the **cluster signature** of $\mathbf{x}_j$. The norm condition implies that exactly one of the components of $\mathbf{z}_j$ is 1; all others are 0. For instance,

if there are $k = 5$ mechanisms (clusters) in the data and $\mathbf{x}_j \in C_4$, then $\mathbf{z}_j = (0, 0, 0, 1, 0)$.[57]

57: This notation can be generalized to **fuzzy clusters**: the cluster signature of $\mathbf{x}_j$ is

$$\mathbf{z}_j \in [0,1]^k, \quad \|\mathbf{z}_j\|_2 = 1;$$

if $\mathbf{z}_j = (0, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)$, say, then we would interpret $\mathbf{x}_j$ as belonging equally to clusters C_3 and C_4 or as having probability $1/2$ of belonging to either C_3 or C_4.

If we write

$$P(\mathbf{z}_j) = \pi_1^{z_{j,1}} \times \cdots \times \pi_k^{z_{j,k}} = \prod_{\ell=1}^{k} \pi_\ell^{z_{j,\ell}}$$

and

$$P(\mathbf{x}_j \mid \mathbf{z}_j) = P(\mathbf{x}_j \mid \boldsymbol{\theta}_1)^{z_{j,1}} \times \cdots \times P(\mathbf{x}_j \mid \boldsymbol{\theta}_k)^{z_{j,k}} = \prod_{\ell=1}^{k} P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell)^{z_{j,\ell}},$$

we recover the **mixture model**:

$$P(\mathbf{x}_j) = \sum_{\ell=1}^{k} \pi_\ell P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell) = \sum_{\ell=1}^{k} P(\mathbf{z}_j \in C_\ell) P(\mathbf{x}_j \mid \mathbf{z}_j \in C_\ell).$$

Generative Process In practice, then, we can imagine that the dataset $\mathbf{X}$ is generated as follows. For $1 \le j \le n$:

1. draw a cluster signature $\mathbf{z}_j \sim \mathcal{G}_k(\boldsymbol{\pi}) = \text{Mult}_k(\boldsymbol{\pi})$, and
2. draw an observation $\mathbf{x}_j$ from the corresponding mechanism according to $P(\mathbf{x}_j \mid \mathbf{z}_j)$.

But we usually do not have access to this **generative process**; instead, we are given $\mathbf{X}$ and the clustering task is to determine how likely it is that component C_ℓ, $1 \le \ell \le k$, is **responsible** for observation $\mathbf{x}_j$, $1 \le j \le n$.

To do so, we need to compute the probabilities

$$\gamma(z_{j,\ell}) = P(\mathbf{z}_j \in C_\ell \mid \mathbf{x}_j), \quad \forall j, \ell.$$

This is difficult to do directly; we use **Bayes' Theorem** to provide an easier handle on the computations:

$$\begin{aligned}\gamma(z_{j,\ell}) = P(\mathbf{z}_j \in C_\ell \mid \mathbf{x}_j) &= \frac{P(\mathbf{z}_j \in C_\ell) P(\mathbf{x}_j \mid \mathbf{z}_j \in C_\ell)}{P(\mathbf{x}_j)} \\ &= \frac{P(\mathbf{z}_j \in C_\ell) P(\mathbf{x}_j \mid \mathbf{z}_j \in C_\ell)}{\sum_{\nu=1}^{k} P(\mathbf{z}_j \in C_\nu) P(\mathbf{x}_j \mid \mathbf{z}_j \in C_\nu)} = \frac{\pi_\ell P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell)}{\sum_{\nu=1}^{k} \pi_\nu P(\mathbf{x}_j \mid \boldsymbol{\theta}_\nu)}.\end{aligned}$$

The **clustering objective** is to infer $\{\pi_\ell\}_{\ell=1}^k$, $\{\boldsymbol{\theta}_\ell\}_{\ell=1}^k$ from $\mathbf{X}$ for a fixed k, to obtain the desired probabilities $\gamma(z_{j,\ell})$.

Denote

$$\boldsymbol{\Theta} = \{\pi_1, \ldots, \pi_k, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_\ell\}.$$

If we further assume that the $\mathbf{x}_j$ are **independently** drawn by the generative process, then, by construction:

$$P(\mathbf{X} \mid \boldsymbol{\Theta}) = \prod_{j=1}^{n} \sum_{\ell=1}^{k} \pi_k P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell),$$

or

$$\text{LL}(\boldsymbol{\Theta}) = \ln P(\mathbf{X} \mid \boldsymbol{\Theta}) = \sum_{j=1}^{n} \ln \left(\sum_{\ell=1}^{k} \pi_k P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell) \right).$$

The **maximum likelihood estimator** (MLE) of $\boldsymbol{\Theta}$ is

$$\boldsymbol{\Theta}_{\text{MLE}} = \arg\max_{\boldsymbol{\Theta}} \left\{ \ln P(\mathbf{X} \mid \boldsymbol{\Theta}) \right\};$$

if we have information about the **prior** $P(\boldsymbol{\Theta})$, then we may use the **maximum a posteriori estimator** (MAP) instead:

$$\boldsymbol{\Theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\Theta}} \left\{ \ln P(\mathbf{X} \mid \boldsymbol{\Theta}) + \ln P(\boldsymbol{\Theta}) \right\}.$$

Whether we use MLE or MAP depend, in large part, on the form taken by the component distributions.

Gaussian Mixture Models A standard assumption is that all clusters are generated by Gaussian mechanisms, which is to say that $P(\mathbf{x}_j \mid \boldsymbol{\theta}_\ell)$ arises from a **multivariate Gaussian** distribution (GMM):

$$\mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell, \boldsymbol{\Sigma}_\ell) = \frac{1}{\sqrt{(2\pi)^p |\boldsymbol{\Sigma}_\ell|}} \exp\left(-\tfrac{1}{2}(\mathbf{x}_j - \boldsymbol{\mu}_\ell)^\top \boldsymbol{\Sigma}_\ell^{-1} (\mathbf{x}_j - \boldsymbol{\mu}_\ell)\right),$$

where $\boldsymbol{\mu}_\ell \in \mathbb{R}^p$ and $\boldsymbol{\Sigma}_\ell$ is a symmetric positive semi-definite matrix. Thus, if there are k components, then

$$P(\mathbf{x}_j \mid \boldsymbol{\Theta}) = \sum_{\ell=1}^{k} \pi_\ell \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell, \boldsymbol{\Sigma}_\ell)$$

and

$$\text{LL}(\boldsymbol{\Theta}) = \ln P(\mathbf{X} \mid \boldsymbol{\Theta}) = \sum_{j=1}^{n} \ln \left(\sum_{\ell=1}^{k} \pi_\ell \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell, \boldsymbol{\Sigma}_\ell) \right).$$

It is straightforward to show that

$$\nabla \text{LL}_{\boldsymbol{\mu}_\ell}(\boldsymbol{\Theta}) = \boldsymbol{\Sigma}_\ell^{-1} \sum_{j=1}^{n} \gamma(z_{j,\ell})(\mathbf{x}_j - \boldsymbol{\mu}_\ell),$$

so that the **MLE estimators** for the **mean vectors** are

$$\hat{\boldsymbol{\mu}}_\ell = \frac{\displaystyle\sum_{j=1}^{n} \gamma(z_{j,\ell})\, \mathbf{x}_j}{\displaystyle\sum_{j=1}^{n} \gamma(z_{j,\ell})}.$$

Thus $\hat{\boldsymbol{\mu}}_\ell$ is a weighted mean of the observations of $\mathbf{X}$, with weights corresponding to the posterior probability $\gamma(z_{j,\ell})$ that the $\ell-$th component was responsible for generating $\mathbf{x}_j$.

Simultaneously, we can show that

$$\nabla \text{LL}_{\boldsymbol{\Sigma}_\ell}(\boldsymbol{\Theta}) = \sum_{j=1}^{n} \frac{\pi_\ell}{P(\mathbf{x}_j \mid \boldsymbol{\Theta})} \cdot \frac{\partial \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell, \boldsymbol{\Sigma}_\ell)}{\partial \boldsymbol{\Sigma}_\ell};$$

slightly more complicated manipulations show that the **MLE estimators**

for the **covariance matrices** are also weighted averages:

$$\hat{\boldsymbol{\Sigma}}_\ell = \frac{\sum_{j=1}^{n} \gamma(z_{j,\ell})(\mathbf{x}_j - \hat{\boldsymbol{\mu}}_\ell)(\mathbf{x}_j - \hat{\boldsymbol{\mu}}_\ell)^\top}{\sum_{j=1}^{n} \gamma(z_{j,\ell})}.$$

Finally, to obtain the mixture probabilities π_ℓ, we must maximize $\text{LL}(\boldsymbol{\Theta})$ with respect to $\boldsymbol{\pi}$, subject to $\pi_\ell \in [0, 1]$ and $\pi_1 + \cdots + \pi_k = 1$; we can use **Lagrange multipliers** to show that the MLE estimates of the mixture probabilities are also an average:

$$\hat{\pi}_\ell = \frac{1}{n} \sum_{j=1}^{n} \gamma(z_{j,\ell}).$$

So we have nice expressions for the MLE estimates $\hat{\boldsymbol{\Theta}}$.[58]

58: There is a problem, however: we need the clustering probabilities $\gamma(z_{j,\ell})$ in order to provide the MLE estimates ... but the former depend on the MLE estimates!

Expectation-Maximization Algorithm While there is no **closed-form solution** allowing us to express the cluster signatures directly in terms of the observed data $\mathbf{X}$, there is a simple iterative solution based on the **Expectation-Maximization** algorithm for GMM.

Input: X
Output: $\boldsymbol{\Theta}^*$ which maximizes $\text{LL}(\boldsymbol{\Theta})$

0. Initialize $\boldsymbol{\Theta}^{[0]} = \left\{\boldsymbol{\mu}_\ell^{[0]}, \boldsymbol{\Sigma}_\ell^{[0]}, \pi_\ell^{[0]}\right\}_{\ell=1}^{k}$ and set

$$\text{LL}^{[0]} = \text{LL}(\boldsymbol{\Theta}^{[0]});$$

For $i = 0$ to `max_step`, do:

1. **E**(xpectation)**-step**: compute the responsibilities

$$\gamma(z_{j,\ell}^{[i]}) = \frac{\pi_\ell^{[i]} \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell^{[i]}, \boldsymbol{\Sigma}_\ell^{[i]})}{\sum_{v=1}^{k} \pi_v^{[i]} \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_v^{[i]}, \boldsymbol{\Sigma}_v^{[i]})}, \quad \forall j, \ell;$$

2. **M**(aximization)**-step**: update the parameters

$$\boldsymbol{\mu}_\ell^{[i+1]} = \frac{\sum_{j=1}^{n} \gamma(z_{j,\ell}^{[i]})\, \mathbf{x}_j}{\sum_{j=1}^{n} \gamma(z_{j,\ell}^{[i]})}, \quad \forall \ell;$$

$$\boldsymbol{\Sigma}_\ell^{[i+1]} = \frac{\sum_{j=1}^{n} \gamma(z_{j,\ell}^{[i]})(\mathbf{x}_j - \boldsymbol{\mu}_\ell^{[i]})(\mathbf{x}_j - \boldsymbol{\mu}_\ell^{[i]})^\top}{\sum_{j=1}^{n} \gamma(z_{j,\ell}^{[i]})}, \quad \forall \ell,$$

$$\pi_\ell^{[i+1]} = \frac{1}{n} \sum_{j=1}^{n} \gamma(z_{j,\ell}^{[i]}), \quad \forall \ell;$$

3. Set $\text{LL}^{[i+1]} = \text{LL}(\boldsymbol{\Theta}^{[i]})$ and check for convergence according to some **convergence criterion**

$$(\|\boldsymbol{\Theta}^{[i]} - \boldsymbol{\Theta}^{[i+1]}\| < \varepsilon, \quad \text{say}) :$$

if satisfied, set $\boldsymbol{\Theta}^* = \boldsymbol{\Theta}^{[i+1]}$; otherwise, set $i := i+1$ and repeat steps 1 to 3.

There are two main limitations to using EM for GMM:

- EM is **costlier** (has a longer run-time) than $k-$means, and depending on the initialization, the algorithm may converge to a **local critical point** which is not necessarily the global maximizer;
- as the algorithm iterates, two (or more) GMM clusters can **collapse** into a single GMM cluster.

Note that the EM algorithm can be sped-up by **first running** $k-$**means** and using the mean vector, covariance matrix, and proportion of observations in the $k-$means cluster C_ℓ for the initialization of $\boldsymbol{\mu}_\ell^{[0]}$, $\boldsymbol{\Sigma}_\ell^{[0]}$, and π_ℓ for $1 \leq \ell \leq k$.

The collapsing of clusters can be mitigated by monitoring $\|\boldsymbol{\Sigma}_\ell^i\|_2$ and randomly resetting $\boldsymbol{\mu}_\ell^{[i]}$, $\boldsymbol{\Sigma}_\ell^{[i]}$ when some threshold is reached.

Special Cases and Variants In a GMM with k components, if $\boldsymbol{\Sigma}_\ell = \boldsymbol{\Sigma} = \sigma^2 \mathbf{I}_n$ for all ℓ, then

$$P(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^p}\sigma} \cdot \exp\left(-\frac{1}{2\sigma^2}\|(\mathbf{x} - \boldsymbol{\mu}_\ell)\|_2^2\right);$$

the EM algorithm applied to this special case leads to

$$\textbf{E-step:} \quad \gamma(z_{j,\ell}^{[i]}) = \frac{\pi_\ell^{[i]} \exp\left(-\|\mathbf{x}_j - \boldsymbol{\mu}_\ell^{[i]}\|_2^2/2\sigma^2\right)}{\sum_{v=1}^k \pi_v^{[i]} \exp\left(-\|\mathbf{x}_j - \boldsymbol{\mu}_v^{[i]}\|_2^2/2\sigma^2\right)}$$

$$\textbf{M-step:} \quad \boldsymbol{\mu}_\ell^{[i+1]} = \frac{\sum_{j=1}^n \gamma(z_{j,\ell}^{[i]})\, \mathbf{x}_j}{\sum_{j=1}^n \gamma(z_{j,\ell}^{[i]})}$$

$$\pi_\ell^{[i+1]} = \frac{1}{n}\sum_{j=1}^n \gamma(z_{j,\ell}^{[i]}).$$

When $\sigma \longrightarrow 0$, we can show that

$$\gamma(z_{j,\ell}) \longrightarrow \begin{cases} 1 & \text{if } \ell = \arg\min_v \left\{\|\mathbf{x}_j - \boldsymbol{\mu}_v\|_2^2\right\} \\ 0 & \text{otherwise} \end{cases}$$

which is simply the formulation for $k-$means. Note that the components do not need to be multivariate Gaussians; there is a **general EM algorithm** that takes advantage of the concavity of the ln function [2].

If the dataset of observations is **binary**, as may occur in image datasets (each pixel taking on the values 0 or 1, depending as to whether the pixel

is white or black, say), we can modify GMM so that $P(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell)$ arises from a **multivariate Bernoulli** distribution:

$$\mathcal{B}(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell) = \prod_{\nu=1}^{p} \mu_{\ell,\nu}^{x_{j,\nu}} (1 - \mu_{\ell,\nu})^{1-x_{j,i}},$$

where $\boldsymbol{\mu}_\ell \in [0,1]^p$. Thus, if there are k components, then

$$P(\mathbf{x}_j \mid \boldsymbol{\Theta}) = \sum_{\ell=1}^{k} \pi_\ell \mathcal{B}(\mathbf{x}_j \mid \boldsymbol{\mu}_\ell)$$

and

$$\mathrm{LL}(\boldsymbol{\Theta}) = \ln P(\mathbf{X} \mid \boldsymbol{\Theta}) = \sum_{j=1}^{n} \ln \left(\sum_{\ell=1}^{k} \pi_\ell \prod_{\nu=1}^{p} \mu_{\ell,\nu}^{x_{j,\nu}} (1 - \mu_{\ell,\nu})^{1-x_{j,i}} \right).$$

We can find $\boldsymbol{\Theta}^*$ that maximizes $\mathrm{LL}(\boldsymbol{\Theta})$ by using the EM algorithm for the Bernoulli Mixture Model (BMM): the EM algorithm applied to this special case leads to

$$\textbf{E-step:} \quad \gamma(z_{j,\ell}^{[i]}) = \pi_\ell^{[i]} \prod_{\nu=1}^{p} \left(\mu_{\ell,\nu}^{[i]}\right)^{x_{j,\nu}} (1 - \mu_{\ell,\nu}^{[i]})^{1-x_{j,i}}$$

$$\textbf{M-step:} \quad \boldsymbol{\mu}_\ell^{[i+1]} = \frac{\sum_{j=1}^{n} \gamma(z_{j,\ell}^{[i]})\, \mathbf{x}_j}{\sum_{j=1}^{n} \gamma(z_{j,\ell}^{[i]})}$$

$$\pi_\ell^{[i+1]} = \frac{1}{n} \sum_{j=1}^{n} \gamma(z_{j,\ell}^{[i]}),$$

with initialization $\pi_\ell^{[0]} = \frac{1}{k}$ and

$$\boldsymbol{\mu}_\ell^{[0]} \sim \prod_{\nu=1}^{p} \mathcal{U}(0.25, 0.75)$$

for $1 \leq \ell \leq k$.

Other variants include **Generalized EM**, **Variational EM**, and **Stochastic EM** [2]. Note that the essence of EM methods remains the same for all algorithms: we attempt to "guess" the value of the "hidden" cluster variable $z_{j,\ell}$ in the **E-step**, and we update the model parameters in the **M-step**, based on the approximated responsibilities found in the $E-$step.

Interestingly, EM can detect overlapping clusters (unlike $k-$means). But most variants share the same limitations: convergence to a global maximizer is not guaranteed; it may be quite slow even when it does converge, and the correct number of components is assumed to be known prior to analysis.

Example: Gapminder Dataset We cluster the 2011 Gapminder dataset using the `mclust` implementation of EM in `R`; no parameters need be specified (unless we want to use a different dissimilarity measure).[59]

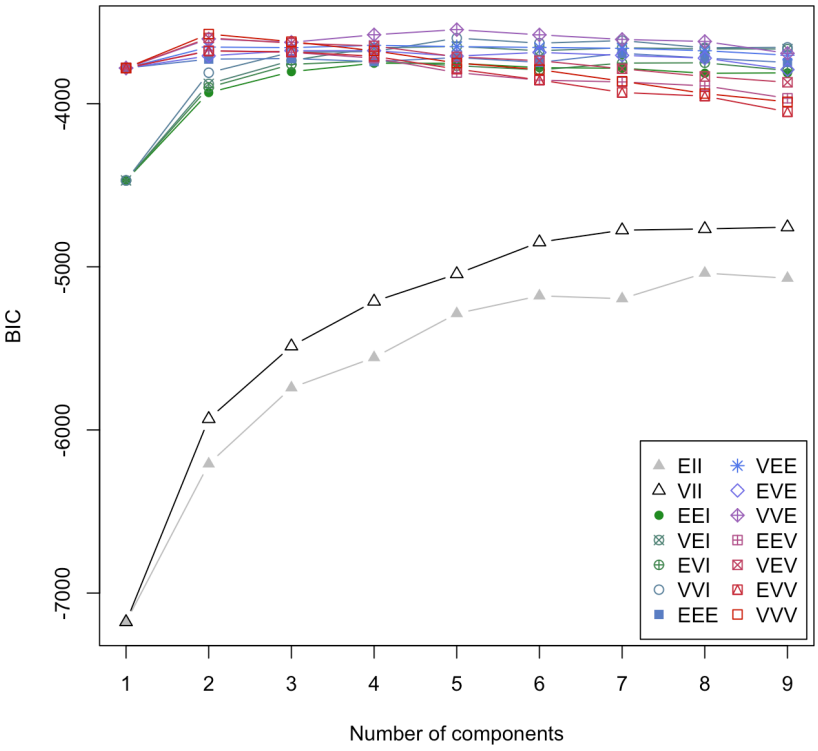
59: The `mclust` vignette contains more information.

This implementation determines the optimal number of clusters using BIC (see Section 20.4.3).

We cluster both the **raw** data and the **scaled** data, to showcase the impact scaling can have.

We start by determining the number of sources in the raw data:

```
library(mclust)
set.seed(0)
BIC <- mclustBIC(gapminder.SoCL.2011[,c(3:7)])
plot(BIC)
```

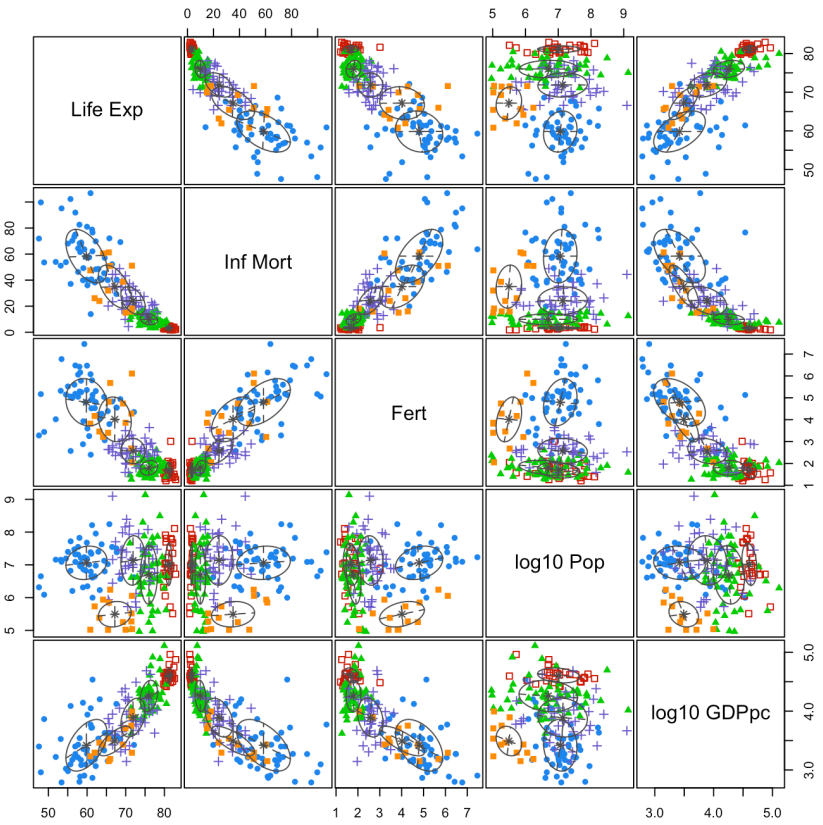



```
summary(BIC)
```

```
Best BIC values:
             VVE,5       VVV,2       VVE,4
BIC      -3545.915 -3573.00451 -3577.50705
BIC diff     0.000   -27.08943   -31.59198
```

This suggests that there are 5 clusters, which we display on the next page.

```
mod1 <- Mclust(gapminder.SoCL.2011[,c(3:7)], x = BIC)
plot(mod1, what = "classification")
```



The MLE estimators for the mean vectors and the covariance matrices for each cluster are computed as below.

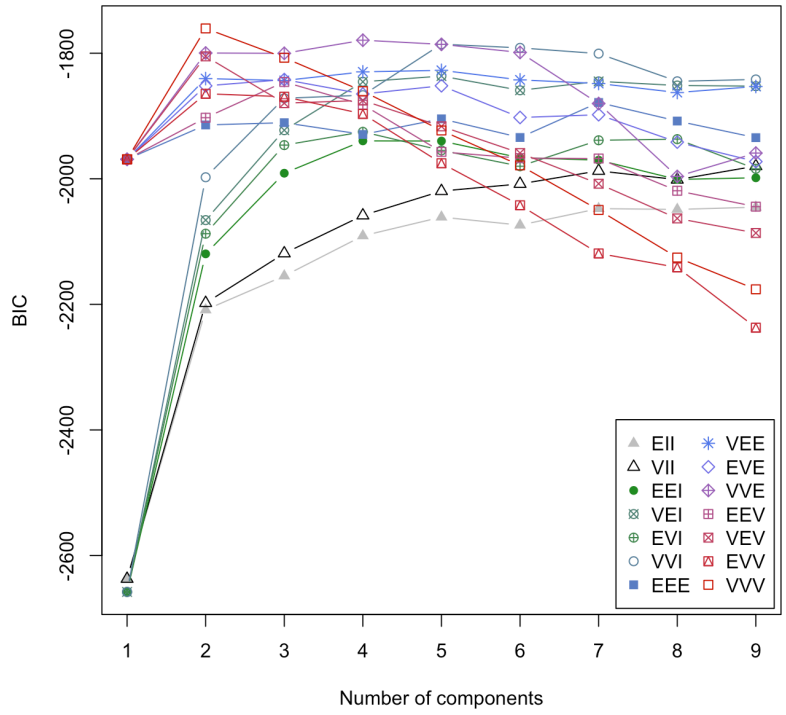
```
summary(mod1, parameters = TRUE)
```

```
Clustering table:
 1  2  3  4  5
51 27 52 41 13

Mixing probabilities:
         1          2          3          4          5
0.28320179 0.14590868 0.27317375 0.22804391 0.06967186

Means:
                [,1]      [,2]      [,3]      [,4]      [,5]
Life Exp    59.834937 81.134281 75.998817 71.888971 67.179530
Inf Mort    58.426776  3.254371  9.913182 24.238438 35.027409
Fert         4.798189  1.691375  1.800049  2.566219  4.014588
log10 Pop    7.061637  7.016423  6.695346  7.133957  5.492939
log10 GDPpc  3.419639  4.606936  4.257569  3.889928  3.491324

Variances:
[,,1]
               Life Exp   Inf Mort       Fert   log10 Pop log10 GDPpc
Life Exp     28.3344610 -55.700455 -0.7444078  0.38472919  1.15550574
Inf Mort    -55.7004554 423.909389 11.8241909  1.11117987 -4.78143371
Fert         -0.7444078  11.824191  1.1663137  0.13060543 -0.21270286
log10 Pop     0.3847292   1.111180  0.1306054  0.26127270 -0.01805256
log10 GDPpc   1.1555057  -4.781434 -0.2127029 -0.01805256  0.19296195
```

```
[,,2]
              Life Exp     Inf Mort        Fert    log10 Pop  log10 GDPpc
Life Exp    0.77045283  0.012104972  0.02774975  0.011407709  0.019766044
Inf Mort    0.01210497  0.679738470  0.01905169  0.003200837 -0.004476642
Fert        0.02774975  0.019051687  0.12771916 -0.042927224 -0.012328032
log10 Pop   0.01140771  0.003200837 -0.04292722  0.416389572 -0.018157675
log10 GDPpc 0.01976604 -0.004476642 -0.01232803 -0.018157675  0.015998859
[,,3]
              Life Exp    Inf Mort         Fert   log10 Pop  log10 GDPpc
Life Exp     4.9940940 -1.72216863  0.142340294  0.10257591  0.137637940
Inf Mort    -1.7221686 17.19416942  0.499861409  0.05750027 -0.180062410
Fert         0.1423403  0.49986141  0.115694181 -0.09747328  0.001937248
log10 Pop    0.1025759  0.05750027 -0.097473276  0.79668779 -0.027754402
log10 GDPpc  0.1376379 -0.18006241  0.001937248 -0.02775440  0.071168246
[,,4]
               Life Exp    Inf Mort        Fert   log10 Pop log10 GDPpc
Life Exp      9.91288695 -14.4994297 -0.09127317  0.15574758  0.36288775
Inf Mort    -14.49942972 112.8663448  3.16128196  0.30356930 -1.26453815
Fert         -0.09127317   3.1612820  0.33684903 -0.03319199 -0.04501104
log10 Pop     0.15574758   0.3035693 -0.03319199  0.57090101 -0.02068610
log10 GDPpc   0.36288775  -1.2645382 -0.04501104 -0.02068610  0.10900700
[,,5]
               Life Exp    Inf Mort       Fert   log10 Pop log10 GDPpc
Life Exp     18.0275970 -35.9634327 -0.5031561  0.24042845  0.74329472
Inf Mort    -35.9634327 273.4386404  7.6150659  0.71441466 -3.08374929
Fert         -0.5031561   7.6150659  1.0499241  0.12825306 -0.19073273
log10 Pop     0.2404284   0.7144147  0.1282531  0.15122047 -0.02103882
log10 GDPpc   0.7432947  -3.0837493 -0.1907327 -0.02103882  0.06307893
```

Let us repeat the procedure on the **scaled** dataset.

```
BIC.s <- mclustBIC(scale(gapminder.SoCL.2011[,c(3:7)]))
plot(BIC.s)
```

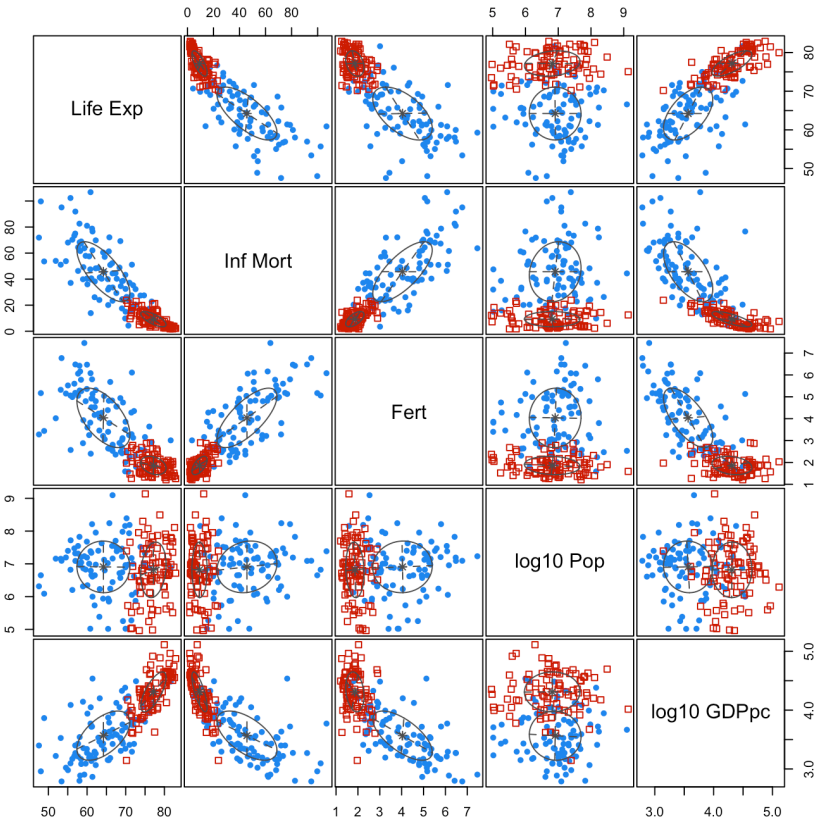


```
summary(BIC.s)
```

```
Best BIC values:
            VVV,2       VVE,4       VVI,5
BIC      -1760.55 -1779.31339 -1785.64061
BIC diff     0.00   -18.76374   -25.09096
```

This suggests that there are 2 clusters, as seen below.

```
mod2 <- Mclust(gapminder.SoCL.2011[,c(3:7)], x = BIC.s)
plot(mod2, what = "classification")
```



The MLE estimators for the mean vectors and covariance matrices for each cluster are computed as below.

```
summary(mod2, parameters = TRUE)
```

```
Clustering table:
 1  2
91 93

Mixing probabilities:
        1         2
0.5059039 0.4940961
```

```
Means:
                [,1]      [,2]
Life Exp    64.226346 77.164028
Inf Mort    45.696249  9.268183
Fert         4.038171  1.860683
log10 Pop    6.906989  6.816294
log10 GDPpc  3.566453  4.310365

Variances:
[,,1]
                Life Exp    Inf Mort        Fert   log10 Pop log10 GDPpc
Life Exp      46.94600319 -112.051411 -5.17027793 -0.06217152  1.55664144
Inf Mort    -112.05141081  531.145010 22.59825954  2.37700481 -5.67438362
Fert          -5.17027793   22.598260  1.84974433  0.03491542 -0.37634844
log10 Pop     -0.06217152    2.377005  0.03491542  0.62342443 -0.01356435
log10 GDPpc    1.55664144   -5.674384 -0.37634844 -0.01356435  0.17508305
[,,2]
               Life Exp     Inf Mort        Fert    log10 Pop  log10 GDPpc
Life Exp     10.7636945 -13.79843667 -0.40969404  0.508375039  0.789463554
Inf Mort    -13.7984367  34.33063934  1.47575086 -0.094412049 -1.497455669
Fert         -0.4096940   1.47575086  0.17525844 -0.037372594 -0.032046730
log10 Pop     0.5083750  -0.09441205 -0.03737259  0.719418611  0.002308201
log10 GDPpc   0.7894636  -1.49745567 -0.03204673  0.002308201  0.115062583
```

22.4.4 Affinity Propagation

Affinity propagation (AP) is a fairly recent arrival on the clustering stage [11, 12]; it takes a somewhat novel perspective on clustering although, as might be expected, there are still similarities to other clustering methods, in particular, DBSCAN and $k-$means.

AP takes the $k-$**medoids** algorithm as a jumping off point. Unlike $k-$means or EM, this algorithm does not operate on statistical principles; rather, it selects existing observations to act as the **exemplar** for a particular cluster (rather than a mean vector, as in $k-$means; see Figure 22.23 for an illustration).

The $k-$mediods algorithm refines the selection of these exemplars so that in the final (stable) configuration, the observations assigned to an exemplar are quite similar to it, relative to other exemplars. As the name suggests, the number of clusters k must be selected prior to running the algorithm; as is the case with $k-$means, $k-$medoids is **non-deterministic** and is sensitive to the **initial choice** of exemplars and similarity metric.

The AP algorithm attempts to overcome the issues arising with $k-$medoids, using Bayesian network theory (in particular, belief propagation networks and factor graphs), and treats observations as a **connected graph**. In this approach, each graph vertex can:

- **communicate** with any other vertex, and
- act as a **possible** exemplar for other observations.

The selection of exemplars is determined by exchanging real-valued **messages** between points. Eventually, sets of exemplars and data points associated with each exemplar are generated from this iterative process, forming clusters. Messages are updated on the basis of fairly simple formulae. As in all clustering contexts, a similarity measure s must first

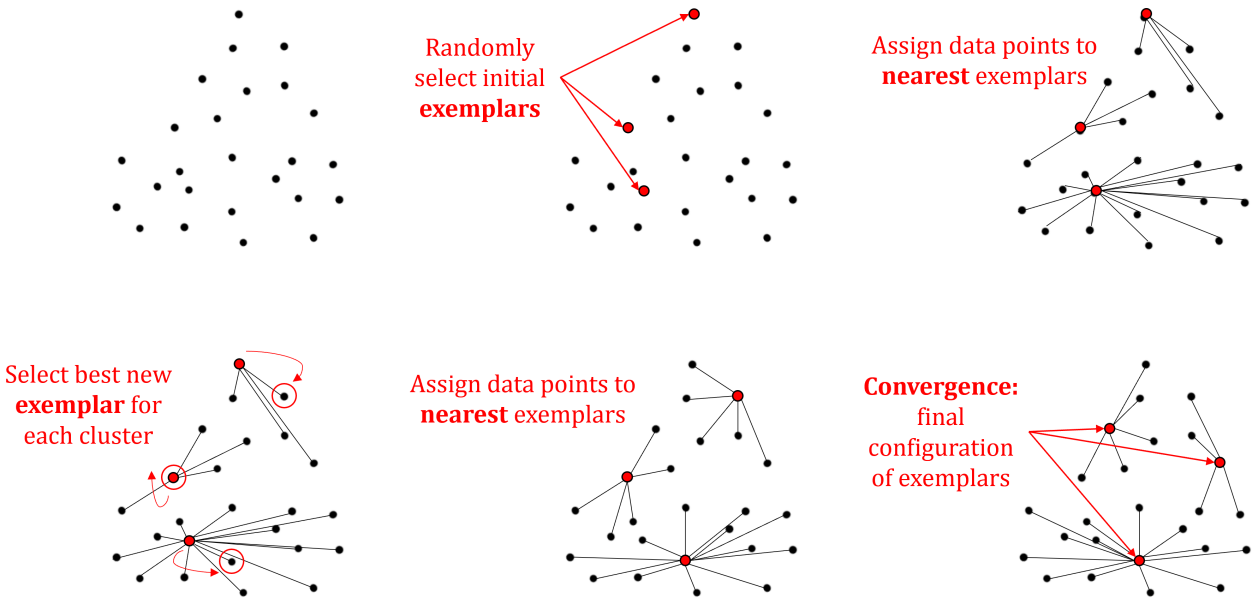


Figure 22.23: Illustration of 3−mediods on an artificial dataset (modified from [11]).

be selected prior to clustering: for distinct pairs (i,k), $s(i,k)$ represents initially the **suitability of k as an exemplar of i** (this suitability will be updated as "messages" are passed between observations).

Each observation k is further assigned a **preference** $s(k,k)$ that it be chosen as an exemplar. The preference can be constant, to indicate no particular initial preference.

Two types of messages get sent:

- the **availability** $a(i,k)$ sent from k to i, which reports on the suitability of k to be an exemplar of i;
- the **responsibility** $r(i,k)$ sent from i to k, which reports on the suitability of i to be represented by k.

The availabilities are initialized to $a(i,k) \leftarrow 0$, the responsibilities to

$$r(i,k) \leftarrow s(i,k) - \max_{k' \neq k} \{a(i,k') + s(i,k')\}.$$

This calculation allows eligible exemplars of an observation to "**compete**" for each observations, in a sense, so they can become that observation's exemplar.[60] After the initial assignment, an availability $a(i,k) = 0$ means that observation i has no affinity for k as its exemplar).

60: As candidate exemplars are themselves observations, we can also compute **self-responsibility**: $r(k,k) \leftarrow s(k,k) - \max_{k \neq k'} \{s(k,k')\}$.

Subsequently, the focus switches back and forth between the exemplar and the observation perspective, with observations looking for available exemplars:

$$a(i,k) \leftarrow \begin{cases} \min\left\{0, r(k,k) + \sum\limits_{i' \notin \{i,k\}} \max\{0, r(i,k)\}\right\} & i \neq k \\ \sum\limits_{i' \neq k} \max\{0, r(i',k)\} & i = k \end{cases}$$

The case $i = k$ is intended to reflect **current evidence** that observation k is an exemplar. The responsibilities and availabilities are updated,

reflecting the **current affinity** that one observation has for choosing another observation as its exemplar (hence the name), until the quantities converge to $r(i,k)$ and $a(i,k)$, respectively, for all pairs of observations (i,k).

This leads to the **cluster assignment** $\{c_1,\ldots,c_n\}$, where

$$c_i = \arg\max_k \{a(i,k) + r(i,k)\}, \quad 1 \leq i \leq n;$$

if i is an observation with associated exemplar k, then $c_i = c_k = k$.

The fact that any observation can become an exemplar when the quantities are updated, and thus that the number of clusters is not an algorithm parameter, is an important distinction between AP and $k-$medoids (and other segmentation clustering approaches). The process is illustrated below.

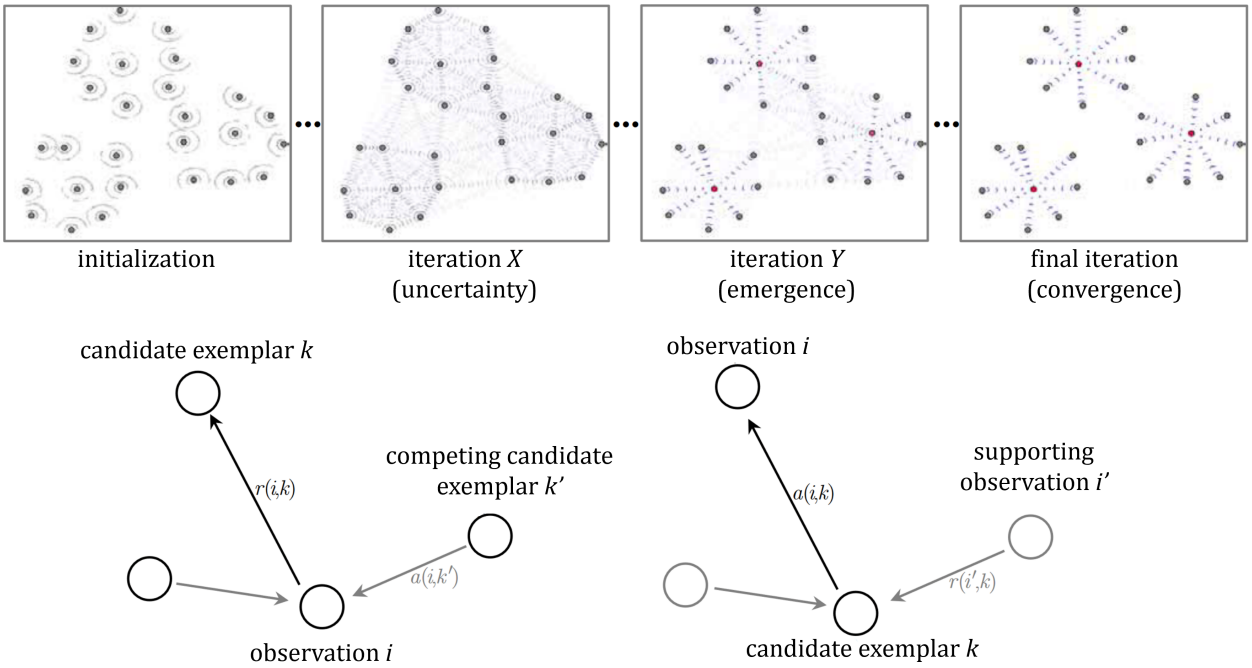


Figure 22.24: Illustration of affinity propagation on an artificial dataset (top); illustration of availability and responsibility (bottom); modified from [11].

Setting Algorithm Parameters Two parameters impact AP's clustering behaviour: the **input preference** (which influences the eventual number of clusters) and the **dampening parameter**.

The input preference determines the suitability of each observation to act as an exemplar; this is often set as the **median similarity** in the data, but it can be tweaked. In principle, certain observations could be assigned preference values in a different manner, perhaps relating to domain knowledge (or previous results).

The **dampening parameter** is slightly more technical. Because affinity propagation creates a directed graph to generate clusters, it can become vulnerable to **graph loops**, which could result in algorithmic oscillations (the algorithm may not converge to a particular solution). The dampening factor acts to control this oscillation problem.

Comparison with Other Algorithms Performance of clustering algorithms can be considered both in general (e.g., based on best/worst cases of an implemented algorithm) or in the context of applications in particular domains – one major **drawback** of AP is the calculation cost of the similarity matrix, which is $O(n^2)$.

Once the similarity matrix has been calculated, the number of scalar computations scales linearly in the number of similarities or quadratically in the number of observations if all possible pairwise similarities are used [11]. In other words, AP is slow on larger datasets.

Arguably, one of the key **advantages** of AP (other than not having to specify the number of clusters up front) is its ability to use any similarity measure. As a result, we do not need to alter the dataset to 'fit' with a distance/similarity framework.[61]

61: Such as by changing categorical variables into numeric variables in some way, or ignoring categorical variables altogether.

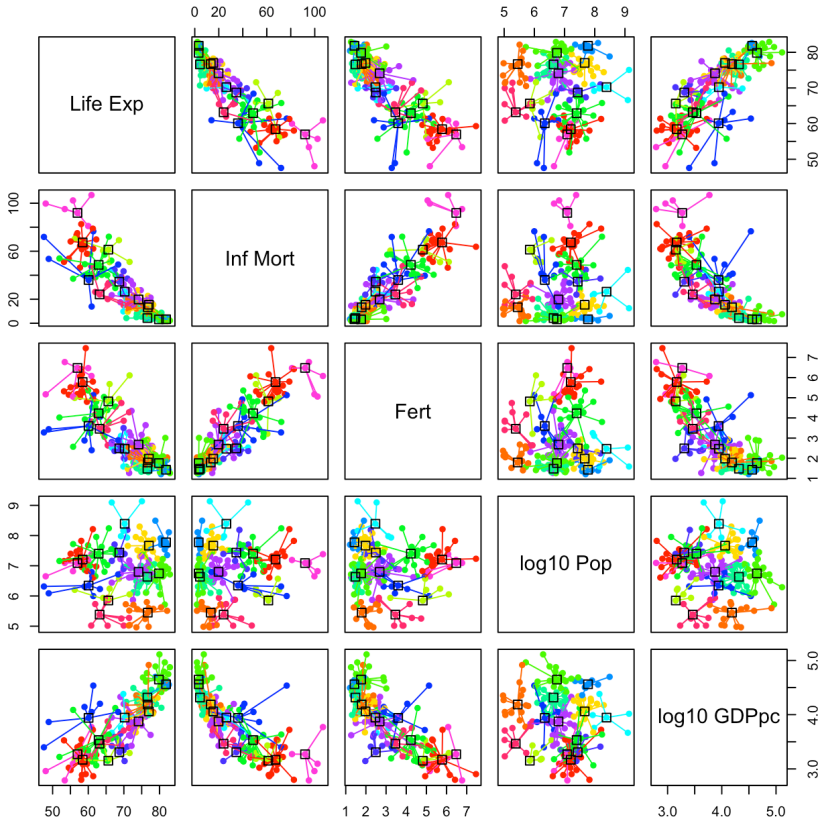
Example We once again re-visit the 2011 (scaled) Gapminder dataset. We use the AP implementation found in the `R` package `apcluster`, with similarity $s(i,k) = -\|\mathbf{x}_i - \mathbf{x}_k\|^2$. We start by setting the input preference as the median similarity and obtain 14 clusters.

```
library(apcluster)
ap.gap.1 <- apcluster(negDistMat(r=2),
    scale(gapminder.SoCL.2011[,c(3:7)]))
ap.gap.1
```

```
Number of clusters    =  14

Exemplars:
   bfa brb col com dnk gha hrv idn ita nam npl pry tcd vut
Clusters:
   Cluster 1, exemplar bfa:
      afg bdi ben bfa civ cmr gin lbr moz mwi ner nga ssd tgo uga zmb cod
   Cluster 2, exemplar brb:
      bhs blz brb cpv isl mdv mlt sur brn lca mne vct atg grd syc
   Cluster 3, exemplar col:
      arg bra chl col dza irn lka mar mex mys per rou tha tur ukr ven vnm
   Cluster 4, exemplar com:
      com dji gmb gnb mrt tls
   Cluster 5, exemplar dnk:
      aut bel che dnk fin grc irl isr lux nld nor nzl omn prt qat sgp swe are
      bhr cyp kwt sau
   Cluster 6, exemplar gha:
      eri eth gha hti ken lao mdg pak png rwa sdn sen tza yem zaf zwe irq
   Cluster 7, exemplar hrv:
      bgr blr cri cub cze est hrv hun lbn ltu lva mus srb svk svn tto ury alb
      mkd bih
   Cluster 8, exemplar idn:
      chn egy idn ind phl rus
   Cluster 9, exemplar ita:
      aus can deu esp fra gbr ita jpn kor pol usa
   Cluster 10, exemplar nam:
      bwa cog gab gnq lso nam swz lby tkm
   Cluster 11, exemplar npl:
      bgd khm mmr npl tjk uzb prk
```

```
   Cluster 12, exemplar pry:
      arm aze bol btn dom ecu geo gtm hnd jam kaz kgz mda mng nic pan pry slv
      tun jor pse syr
   Cluster 13, exemplar tcd:
      ago caf mli sle som tcd
   Cluster 14, exemplar vut:
      fji guy stp slb ton vut wsm kir fsm
```

```
plot(ap.gap.1, gapminder.SoCL.2011[,c(3:7)])
```



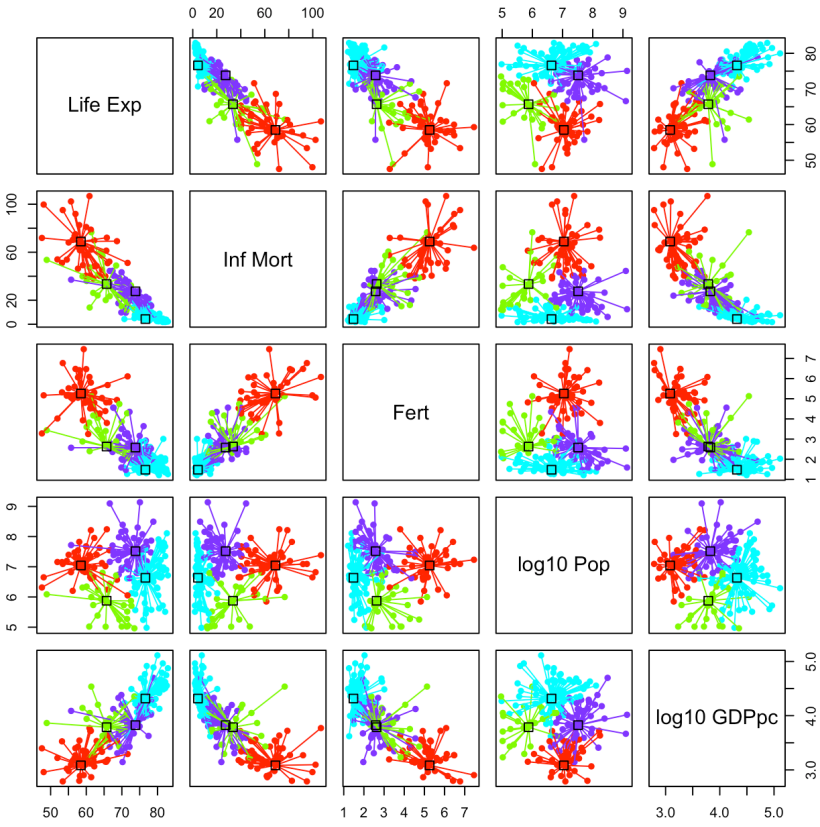
If instead we use the minimum similarity, we obtain 4 clusters (**exemplars:** Guinea, Guyana, Croatia, Morocco).

```
ap.gap.2 <- apcluster(negDistMat(r=2),
    scale(gapminder.SoCL.2011[,c(3:7)]), q=0)
ap.gap.2
```

```
Number of clusters    =  4

Exemplars:
   gin guy hrv mar
Clusters:
   Cluster 1, exemplar gin:
      afg ago bdi ben bfa caf civ cmr cog com eri eth gha gin gmb gnb hti ken
      lbr lso mdg mli moz mrt mwi ner nga pak png rwa sdn sen sle som ssd tcd
      tgo tza uga zmb zwe cod tls
```

```
Cluster 2, exemplar guy:
   blz btn bwa cpv dji fji gab gnq guy lao mng nam stp sur swz lby slb tkm
   ton vct vut wsm grd kir syc fsm
Cluster 3, exemplar hrv:
   arm aus aut bel bgr bhs blr brb can che chl cri cub cze deu dnk esp est
   fin fra gbr geo grc hrv hun irl isl isr ita jam jpn kor lbn ltu lux lva
   mda mdv mlt mus mys nld nor nzl omn pan pol prt qat rou sgp srb svk svn
   swe tto tun ury alb are bhr brn cyp kwt lca mkd mne sau bih atg
Cluster 4, exemplar mar:
   arg aze bgd bol bra chn col dom dza ecu egy gtm hnd idn ind irn kaz kgz
   khm lka mar mex mmr nic npl per phl pry rus slv tha tjk tur ukr usa uzb
   ven vnm yem zaf irq jor pse syr prk
```

```
plot(ap.gap.2, gapminder.SoCL.2011[,c(3:7)])
```



Which of these two schemes seems to provide a better segmentation?

22.4.5 Fuzzy Clustering

Fuzzy clustering (FC) is also called "soft" clustering (in opposition to "hard" clustering). Rather than assigning each observation to a cluster, they are assigned a **cluster signature**, a set of values that indicate their relative membership in each of the clusters.

The signature vector is often interpreted as a **probability vector**: observation $\mathbf{x}_i$ belongs to cluster ℓ with probability $p_{i,\ell} \geq 0$, with

$$p_{i,1} + \cdots + p_{i,c} = 1, \quad \text{for all } 1 \leq i \leq n.$$

Fuzzy $c-$Means: the Typical Approach The most prevalent algorithm for carrying out FC is called fuzzy $c-$means (FCM). It is a variant of $k-$means with two modifications:

- the presence of a new parameter $m > 1$, called the **fuzzyfier**, which determines the degree of "fuzziness" of the clusters, and
- cluster membership is output as a **weight vector**, with weights in $[0, 1]$ adding to 1.

As in $k-$means, c observations are selected randomly as the initial cluster centroids, as are the membership weights of each observation. The membership weights of each observations, relative to the current centroid, are re-calculated based on how "close" the point is to the given centroid in comparison to the distance to all of the other centroids.[62]

62: The centroid of the ℓth cluster is the weighted average of ALL observations by the degree to which they belong to cluster ℓ.

Effectively, we look for clusters that minimize the objective function

$$\sum_{\ell=1}^{c} \sum_{\mathbf{x}_i \in C_\ell} u_{i,\ell}^m \text{variation}(\mathbf{x}_i, \boldsymbol{\mu}_\ell),$$

where the **degree** $u_{i,\ell}^m$ to which observation $\mathbf{x}_i$ belongs to cluster C_ℓ is

$$u_{i,\ell}^m = \frac{1}{\sum_{j=1}^{c} \left(\frac{\text{variation}(\mathbf{x}_i, \boldsymbol{\mu}_\ell)}{\text{variation}(\mathbf{x}_i, \boldsymbol{\mu}_j)} \right)^{2/(m-1)}}.$$

The value of m effectively determines the width of **fuzziness bands** around clusters, where clusters may overlap with other clusters. Within these bands, if there are overlaps, points will have weights between 0 and 1.

Outside of these bands, points will have a membership of 1 for a particular cluster (that it is close to) and a membership of 0 for other bands.

As with $k-$means, the algorithm is generally run until the change in membership values, or in this case the weights, falls below a particular threshold. In practice, we typically use $m = 2$ and

$$\text{variation}(\mathbf{x}_i, \boldsymbol{\mu}_\ell) = \|\mathbf{x}_i - \boldsymbol{\mu}_\ell\|^2.$$

As $m \to 1$, FCM converges to $k-$means.

Comparison Between Fuzzy $c-$Means and $k-$Means To gain an appreciation for how FCM works, it can be useful to compare its results to those provided by $k-$means. Figure 22.25 shows the same dataset clustered by $k-$means (left) and fuzzy $c-$means (right) [6].

On the right, we can see observations that "belong" to the 2 clusters. FCM is useful in this context because it would seem almost arbitrary for some of the points to be assigned to one or the other cluster (which is what $k-$means does).

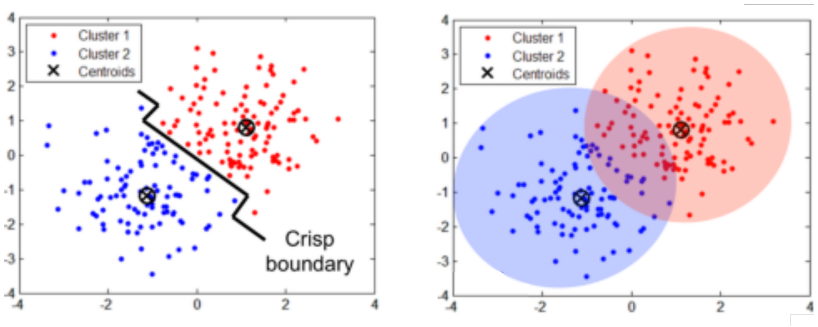


Figure 22.25: Fuzzy $c-$means vs. $k-$means clustering (author unknown).

Other Fuzzy Clustering Options Although FCM is the most popular fuzzy clustering algorithm, it is not a particularly **nuanced** algorithm. Like $k-$means, the resulting clusters are essentially blob-shaped. Sophisticated results can be gained by using more complex algorithms.

The **Gustafson–Kessel** (GK) clustering algorithm [14] is an early extension of FCM which replaces the simple distance measure used in FCM with a (covariance) matrix. This brings FCM more in-line with EM clustering, which also provides fuzzy results, and can be carried out with a variety of statistical models, resulting in a more mature clustering results, albeit at the cost of heavier processing. **FANNY** [22] is another fuzzy approach; it is less sensitive to outliers than FCM is.

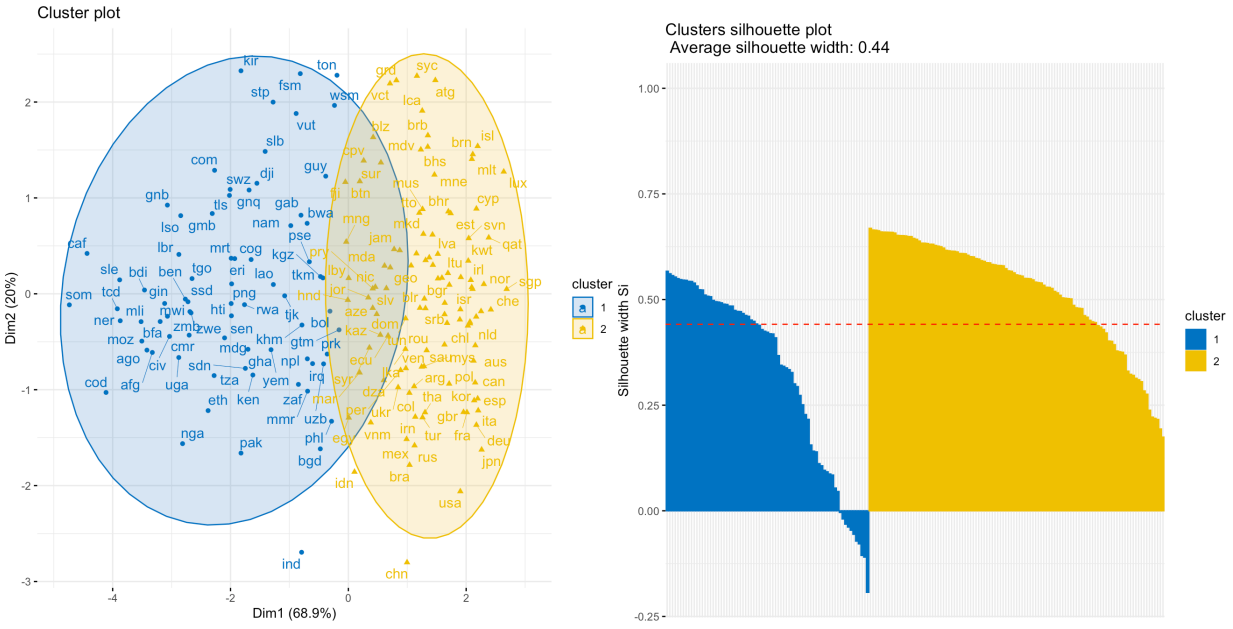
Fuzzy Clustering Validation As with hard clustering, it is important to **validate** fuzzy clusters. A number of validation strategies have been developed; the **Xie-Beni index** is a popular choice. It can be calculated for non-fuzzy clusters as well as for fuzzy clusters. However, it takes into accounts the weights of the points for each clustering by weighting the clustering separation and compactness measures using the membership matrix (i.e., the matrix that contains the weights for each observation with respect to each cluster). Other metrics include the **Tang index** and the **Kwon index** [24, 41, 50].

Example We show some results of FANNY (with $c = 2, 3, 4$, and 6 clusters, implemented in `cluster`'s `fanny()`) and FCM (with $c = 4$ clusters, implemented in `e1071`'s `cmeans()`) on the (scaled) 2011 Gapminder dataset (again, using Euclidean dissimilarity).

We start with FANNY(2).

```
set.seed(987) # for replicability
fuzzy.gap <- cluster::fanny(scale(gapminder.SoCL.2011[,c(3:7)]),
                            k=2, metric="euclidean", maxit=20000)
attributes(fuzzy.gap)

factoextra::fviz_cluster(fuzzy.gap, ellipse.type = "norm", repel = TRUE,
             palette = "jco", ggtheme = theme_minimal(),
             legend = "right")

factoextra::fviz_silhouette(fuzzy.gap, palette = "jco",
                ggtheme = theme_minimal())
```

```
$names
 [1] "membership"  "coeff"       "memb.exp"    "clustering"  "k.crisp"
 [6] "objective"   "convergence" "diss"        "call"        "silinfo"
[11] "data"

$class
[1] "fanny"     "partition"

  cluster size ave.sil.width
1       1   75          0.32
2       2  109          0.52
```



The plots for FANNY(3), FANNY(4), and FANNY(5), are displayed in Figure 22.26.[63]

63: We simply replace `k=2` by `k=3`, `k=4`, and `k=6` in the call to `fanny()`.

```
FANNY(3)
  cluster size ave.sil.width
1       1   64          0.26
2       2   74          0.34
3       3   46          0.15

FANNY(4)
  cluster size ave.sil.width
1       1   53          0.33
2       2   80         -0.01
3       3   50         -0.17
4       4    1          0.00

FANNY(5)
  cluster size ave.sil.width
1       1   30          0.35
2       2   83          0.06
3       3   34         -0.21
4       4   34          0.03
5       5    3          0.60
```

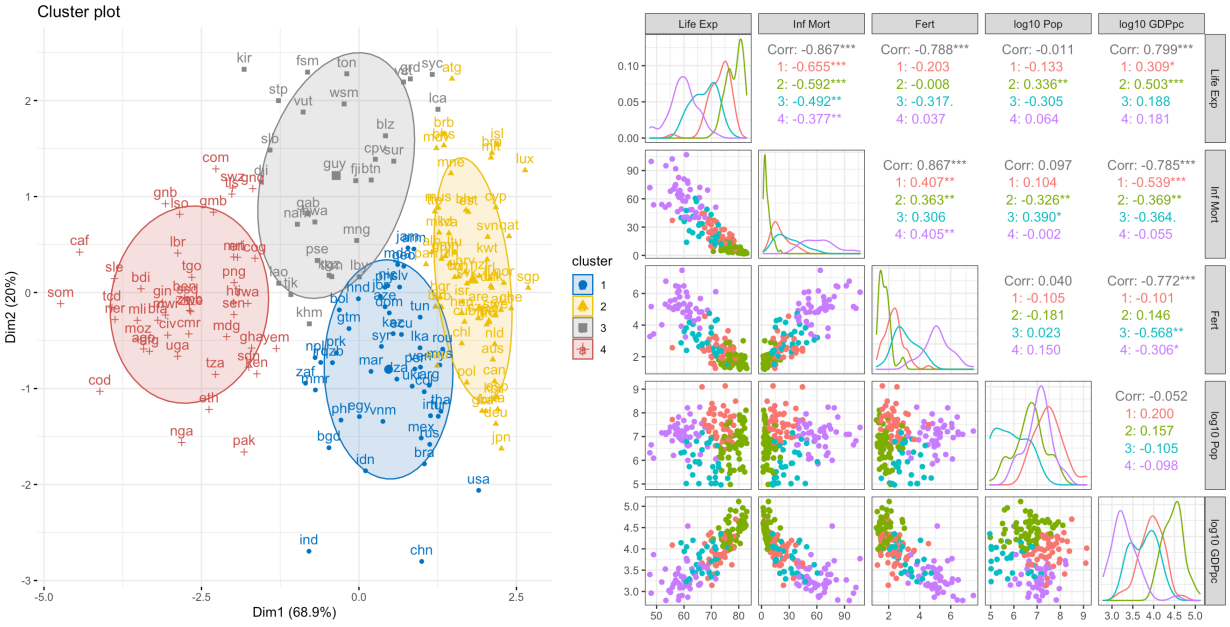
Figure 22.26: Clusters and silhouette profiles for the 2011 Gampinder dataset; FANNY(3) (top row), FANNY(4) (middle row), FANNY(5) (bottom row). The scatterplots are displayed on the data's first two principal components.

The corresponding results for F4M are computed below.

```
cm.gap <- e1071::cmeans(scale(gapminder.SoCL.2011[,c(3:7)]), 4)
attributes(cm.gap)
```

```
$names
[1] "centers"     "size"        "cluster"     "membership"  "iter"
[6] "withinerror" "call"

$class
[1] "fclust"
```

```
factoextra::fviz_cluster(list(data = scale(gapminder.SoCL.2011[,c(3:7)]),
                              cluster=cm.gap$cluster),
             ellipse.type = "norm",
             ellipse.level = 0.68,
             palette = "jco",
             ggtheme = theme_minimal())

GGally::ggpairs(gapminder.SoCL.2011[,c(3:7)],
                ggplot2::aes(color=as.factor(cm.gap$cluster)),
                diag=list(continuous=my_dens))
```



How does that compare to all the other approaches we have used so far? Based on those, how many clusters do you think there are in the 2011 Gapminder dataset?

22.4.6 Cluster Ensembles

We have seen that the choice of clustering method and algorithm parameters may have an impact on the nature and number of clusters in the data; quite often, the resulting clusters are **volatile**. This is aligned with the idea that the ability to accurately assess the quality of a clustering outcome remains **elusive**, for the most part.

The goal of **ensemble clustering** is to combine the results of multiple clustering runs to create a more **robust** outcome.

Most ensemble models use the following two steps to generate an outcome:

1. **generate** different clustering schemes, using different models, parameters, or data selection mechanisms (the **ensemble components**), and
2. **combine** the different results into a single outcome.

Selecting Different Ensemble Components The ensemble components are either **model-based** or **data selection-based**.

In **model-based ensembles**, the different components of the ensemble reflect different models, such as the use of

- different clustering **approaches**;
- different **parameter settings** for a given approach;
- different **randomizations** (for stochastic algorithms),
- or some **combination** of these.

For instance, an ensemble's components could be built from:

1. 5 runs of $k-$means for each of $k = 2, \ldots, 10$, for each of the Euclidean and Manhattan similarities (90 components);
2. the hierarchical clustering outcome for each of the complete, single, average, centroid, and Ward linkage, for each of the Euclidean and Manhattan distances, for each of $k = 2, \ldots, 10$ clusters (90 components);
3. the DBSCAN outcome for each of 5 values of ε^*, for each of $minPts = 2, \ldots, 10$, for each of the Euclidean and Manhattan distances (90 components), and
4. the spectral clustering outcome for each of 3 threshold values τ, for each of the 3 types of Laplacians, for $k = 2, 4, 6, 8, 10$, for each of the Euclidean and Manhattan distances (90 components),

This provides a total of $4 \times 90 = 360$ components. Note that we could also pick algorithms, settings, and similarity measures randomly, from a list of reasonable options.

In **data selection-based ensembles**, we might select a specific clustering approach, combined with a set of parameters, and a given randomization (if the approach is stochastic) and instead build the different components of the model by running the algorithm on different subsets of the data, either *via*:

- selecting **subsets of observations** using random or other probabilistic sampling scheme;
- selecting **subsets of variables**, again using probabilistic sampling, or
- some **combination** of both.

For instance, an ensemble's components could be built using affinity propagation with Euclidean distance and a specific combination of input preference and dampening parameter, and 360 subsets of the data, obtained as follows:

1. for each component, draw a % of observations to sample and a # of variables to select from the data;
2. randomly select a subset with these properties;
3. run affinity propagation on the subset to obtain a clustering outcome.

We could also combine model-based and data selection-based approaches to create the components.

Combining Different Ensemble Components However the components are obtained, we need to find a way to combine them to obtain a **robust clustering consensus**. There are three basic methods to do this:

- **general affiliation**;
- **hypergraph partitioning**, and
- **meta-clustering**.

In the **general affiliation** approach, we consider each pair of observations and determine how frequently they are found in the same clusters in each of the ensemble components. The corresponding proportions create a **similarity matrix**, which can then be used to cluster the data using some graph-based method, such as DBSCAN.

In the **hypergraph partitioning** approach, each observation in the data is represented by a **hypergraph vertex**. A cluster in any of the ensemble components is represented as a **hypergraph hyperedge**, a generalization of the notion of edge which connects (potentially) more than two vertices in the form of a **complete clique**. This hypergraph is then partitioned using graph clustering methods.[64]

The **meta-clustering** approach is also a graph-based approach, except that vertices are associated with each cluster in the ensemble components; each vertex therefore represents a set of **data objects**. A graph partitioning algorithm is then applied to this graph.[65] **Balancing constraints** may be added to the meta-clustering phase to ensure that the resulting clusters are balanced.

Cluster ensembles are implemented in `R` *via* the packages `diceR` and `clue`. More information is available in [2, 1, 45].

64: One major challenge with hypergraph partitioning is that a hyperedge can be "broken" by a partitioning in many different ways, not all of which are qualitatively equivalent. Most hypergraph partitioning algorithms use a constant penalty for breaking a hyperedge.

65: The distribution of the membership of different instances to the meta-partitions can be used to determine its meta-cluster membership, or soft assignment probability.

22.5 Exercises

1. Complete the Ward D linkage, maximum dissimilarity hierarchical clustering results for the 2011 data from `gapminder_all.csv`.
2. Cluster the Iowa Housing, Vowel, Wisconsin Breast Cancer, and Wine datasets using $k-$means, for various distance metrics and algorithm parameters. What is your best estimation for the number of clusters in each case? Validate your results.
3. Cluster the Iowa Housing, Vowel, Wisconsin Breast Cancer, and Wine datasets using hierarchical clustering, for various algorithm parameters. Validate your results.
4. Cluster the Iowa Housing, Vowel, Wisconsin Breast Cancer, and Wine datasets using DBSCAN, for various algorithm parameters. Validate your results.
5. Cluster the Iowa Housing, Vowel, Wisconsin Breast Cancer, and Wine datasets using spectral clustering, for various algorithm parameters. Validate your results.
6. Cluster the Iowa Housing, Vowel, Wisconsin Breast Cancer, and Wine datasets using expectation-maximization clustering, for various algorithm parameters. Validate your results.
7. Cluster the Iowa Housing, Vowel, Wisconsin Breast Cancer, and Wine datasets using affinity propagation clustering, for various algorithm parameters. Validate your results.
8. Cluster the Iowa Housing, Vowel, Wisconsin Breast Cancer, and Wine datasets using fuzzy clustering, for various algorithm parameters. Validate your results.
9. Cluster the Iowa Housing, Vowel, Wisconsin Breast Cancer, and Wine datasets using the combined results of problems 2 to 8. Validate your results.
10. Cluster the datasets
    - `GlobalCitiesPBI.csv`
    - `2016collisionsfinal.csv`
    - `polls_us_election_2016.csv`
    - `HR_2016_Census_simple.xlsx`
    - `UniversalBank.csv`.

    and/or any other datasets of interest, using the approaches discussed in this module (or other other appropriate approaches). Validate your results. Where are there difficulties? What decisions must you make along the way? How could you use the results?

Chapter References

[1] C.C. Aggarwal. *Data Mining: the Textbook*. Cham: Springer, 2015.

[2] C.C. Aggarwal and C.K. Reddy, eds. *Data Clustering: Algorithms and Applications*. CRC Press, 2014.

[3] Enrique Amigó et al. 'A comparison of extrinsic clustering evaluation metrics based on formal constraints'. In: *Inf. Retr.* 12.5 (2009), p. 613.

[4] Renato Cordeiro Amorim. 'Feature Relevance in Ward's Hierarchical Clustering Using the Lp Norm'. In: *J. Classif.* 32.1 (Apr. 2015), pp. 46–62. DOI: 10.1007/s00357-015-9167-1.

[5] F.R. Bach and M.I. Jordan. 'Learning Spectral Clustering, With Application To Speech Separation'. In: *J. Mach. Learn. Res.* 7 (Dec. 2006), pp. 1963–2001.

[6] A. Bagnaro, F. Baltar, and G. Brownstein. 'Reducing the arbitrary: fuzzy detection of microbial ecotones and ecosystems – focus on the pelagic environment'. In: *Environmental Microbiome* 15.16 (2020).

[7] Yoshua Bengio, Pascal Vincent, and Jean-François Paiement. *Learning Eigenfunctions of Similarity: Linking Spectral Clustering and Kernel PCA*. Tech. rep. 1232. Département d'informatique et recherche opérationnelle, Université de Montréal, 2003.

[8] J. Cranshaw et al. 'The Livehoods Project: Utilizing Social Media to Understand the Dynamics of a City'. In: *ICWSM*. Ed. by John G. Breslin et al. The AAAI Press, 2012.

[9] J. d'Huy. 'Scientists Trace Society's Myths to Primordial Origins'. In: *Scientific American (Online)* (Sept. 2016).

[10] B. Desgraupes. *clusterCrit: Clustering Indices* . R package version 1.2.8. 2018.

[11] Delbert Dueck. 'Affinity Propagation: Clustering Data by Passing Messages'. In: *Ph.D. Thesis* (Jan. 2009).

[12] Brendan Frey and Delbert Dueck. 'Clustering by Passing Messages Between Data Points'. In: *Science (New York, N.Y.)* 315 (Mar. 2007), pp. 972–6. DOI: 10.1126/science.1136800.

[13] Maarten Grootendorst. '9 Distance Measures in Data Science '. In: *Towards Data Science* (Feb. 2021).

[14] Donald Gustafson and William Clark Kessel. 'Fuzzy clustering with a fuzzy covariance matrix'. In: *1978 IEEE Conference on Decision and Control including the 17th Symposium on Adaptive Processes* (1978), pp. 761–766.

[15] U. Habib, K. Hayat, and G. Zucker. 'Complex building's energy system operation patterns analysis using bag of words representation with hierarchical clustering'. In: *Complex Adapt. Syst. Model.* 4 (2016), p. 8. DOI: 10.1186/s40294-016-0020-0.

[16] Mahmoud Harmouch. '17 types of similarity and dissimilarity measures used in data science '. In: *Towards Data Science* (Mar. 2021).

[17] N. Harris. *Visualizing DBSCAN Clustering* .

[18] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* , *2nd ed.* Springer, 2008.

[19] Sungsoon Hwang and Jean-Claude Thill. 'Delineating urban housing submarkets with fuzzy clustering'. In: *Environment and Planning B: Planning and Design* 36 (Sept. 2009), pp. 865–882. DOI: 10.1068/b34111t.

[20] G. James et al. *An Introduction to Statistical Learning: With Applications in R* . Springer, 2014.

[21] A. Jawad, K. Kersting, and N. Andrienko. 'Where Traffic Meets DNA: Mobility Mining Using Biological Sequence Analysis Revisited'. In: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS '11. Chicago, Illinois: Association for Computing Machinery, 2011, pp. 357–360. DOI: 10.1145/2093973.2094022.

[22] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis* . John Wiley, 1990.

[23] H. T. Kung and D. Vlah. 'A Spectral Clustering Approach to Validating Sensors via Their Peers in Distributed Sensor Networks'. In: *Int. J. Sen. Netw.* 8.3/4 (Oct. 2010), pp. 202–208. DOI: 10.1504/IJSNET.2010.036195.

[24] Soon Hak Kwon, Jihong Kim, and Seo Ho Son. 'Improved cluster validity index for fuzzy clustering'. In: *Electronics Letters* 57.21 (2021), pp. 792–794.

[25] Tilman Lange et al. 'Stability-Based Model Selection'. In: *Advances in Neural Information Processing Systems (NIPS 2002): 2002* (June 2003).

[26] Joshua M. Lewis, Margareta Ackerman, and Virginia R. de Sa. 'Human Cluster Evaluation and Formal Quality Measures: A Comparative Study'. In: *Cognitive Science* 34 (2012).

[27] Ulrike von Luxburg. 'Clustering Stability: An Overview'. In: *Foundations and Trends in Machine Learning* 2.3 (2010), pp. 235–274. DOI: 10.1561/2200000008.

[28] Ulrike von Luxburg. 'A Tutorial on Spectral Clustering '. In: *Stat. Comput.* 17.4 (2007), pp. 395–416.

[29] Yetis Murat and Ziya Cakici. 'An Integration of Different Computing Approaches in Traffic Safety Analysis'. In: *Transportation Research Procedia* 22 (Dec. 2017), pp. 265–274. DOI: 10.1016/j.trpro.2017.03.033.

[30] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. 'On Spectral Clustering: Analysis and an Algorithm'. In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. NIPS'01. Vancouver, British Columbia, Canada: MIT Press, 2001, pp. 849–856.

[31] Jifeng Ning et al. 'Interactive image segmentation by maximal similarity based region merging'. In: *Pattern Recognition* 43 (Feb. 2010), pp. 445–456. DOI: 10.1016/j.patcog.2009.03.004.

[32] M. Orlowska et al. 'A Comparison of Antioxidant, Antibacterial, and Anticancer Activity of the Selected Thyme Species by Means of Hierarchical Clustering and Principal Component Analysis'. In: *Acta Chromatographica Acta Chromatographica* 28.2 (2016), pp. 207–221. DOI: 10.1556/achrom.28.2016.2.7.

[33] V. U. Panchami and N. Radhika. 'A novel approach for predicting the length of hospital stay with DBSCAN and supervised classification algorithms'. In: *ICADIWT*. IEEE, 2014, pp. 207–212.

[34] V. U. Panchami and N. Radhika. 'A novel approach for predicting the length of hospital stay with DBSCAN and supervised classification algorithms'. In: *ICADIWT*. IEEE, 2014, pp. 207–212.

[35] C. Plant et al. 'Automated detection of brain atrophy patterns based on MRI for the prediction of Alzheimer's disease'. In: *NeuroImage* 50.1 (2010), pp. 162–174.

[36] H. Rosling. *The Health and Wealth of Nations*. Gapminder Foundation, 2012.

[37] G. Schoier and G. Borruso. 'Individual Movements and Geographical Data Mining. Clustering Algorithms for Highlighting Hotspots in Personal Navigation Routes'. In: *Computational Science and Its Applications - ICCSA 2011*. Ed. by Beniamino Murgante et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 454–465.

[38] E. Schubert et al. 'DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN'. In: *ACM Trans. Database Syst.* 42.3 (July 2017). DOI: 10.1145/3068335.

[39] Jiuh-Biing Sheu. 'An Emergency Logistics Distribution Approach for Quick Response to Urgent Relief Demand in Disasters'. In: *Transportation Research Part E: Logistics and Transportation Review* 43 (Nov. 2007), pp. 687–709. DOI: 10.1016/j.tre.2006.04.004.

[40] Wei Sun, Yujun He, and Hong Chang. 'Regional characteristics of CO2 emissions from China's power generation: Affinity propagation and refined Laspeyres decomposition'. In: *International Journal of Global Warming* 11 (Jan. 2017), p. 38. DOI: 10.1504/IJGW.2017.080989.

[41] Yuangang Tang, Fuchun Sun, and Zengqi Sun. 'Improved validation index for fuzzy clustering'. In: *Proceedings of the 2005, American Control Conference, 2005.* IEEE. 2005, pp. 1120–1125.

[42] Osamu Tominaga et al. 'Modeling of Consumers' Preferences for Regular Coffee Samples and Its Application to Product Design'. In: *Food Science and Technology Research* 8 (Aug. 2002), pp. 281–285. DOI: 10.3136/fstr.8.281.

[43] Frederick Tung, Alexander Wong, and David A. Clausi. 'Enabling scalable spectral clustering for image segmentation'. In: *Pattern Recognition* 43.12 (2010), pp. 4069–4076. DOI: https://doi.org/10.1016/j.patcog.2010.06.015.

[44] Lucas Vendramin, Ricardo J. G. B. Campello, and Eduardo R. Hruschka. 'Relative clustering validity criteria: A comparative overview'. In: *Stat. Anal. Data Min.* 3 (2010), pp. 209–235.

[45] Nguyen Xuan Vinh, Julien Epps, and James Bailey. 'Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance'. In: *J. Mach. Learn. Res.* 11 (Dec. 2010), pp. 2837–2854.

[46] Ludi Wang et al. 'Clustering ECG Heartbeat Using Improved Semi-Supervised Affinity Propagation'. In: *IET Software* 11 (June 2017). DOI: 10.1049/iet-sen.2016.0261.

[47] *Ward's Method*.

[48] Wikipedia. *Cluster Analysis Algorithms*.

[49] D.H. Wolpert and W.G. Macready. 'No free lunch theorems for optimization'. In: *IEEE Transactions on Evolutionary Computation* (1997).

[50] X.L. Xie and G. Beni. 'A validity measure for fuzzy clustering'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.8 (1991), pp. 841–847.

[51] Rahul Yedida. *Evaluating Clusters*. 2019.

[52] Lihi Zelnik-Manor and Pietro Perona. 'Self-Tuning Spectral Clustering'. In: *Advances in Neural Information Processing Systems*. Ed. by L. Saul, Y. Weiss, and L. Bottou. Vol. 17. MIT Press, 2005.

Feature Selection and Dimension Reduction

23

by **Patrick Boily**, with contributions from **Olivier Leduc**, **Andrew Macfie**, **Aditya Maheshwari**, and **Maia Pelletier**

Data mining is the collection of processes by which we can extract **useful insights** from data. Hidden in this definition is the idea of **data reduction**: useful insights (whether in the form of summaries, sentiment analyses, and so on) should be "smaller" and "more organized" than the original raw data.

But the challenges presented by high data dimensionality (the so-called **curse of dimensionality**) must be addressed in order to achieve insightful and interpretable analytical results.

In this chapter, we introduce the basic principles of **dimensionality reduction** and a number of **feature selection** methods (filter, wrapper, regularization); we also discuss related advanced topics (SVD, spectral feature selection, UMAP).

23.1 Data Reduction for Insight

For small datasets, the benefits of data mining may not always be evident. Consider, for instance, the following excerpt from a lawn mowing instruction manual (which we consider to be data for the time being):

> Before starting your mower inspect it carefully to ensure that there are no loose parts and that it is in good working order.

It is a **short** and **organized** way to convey a message. It could be further shortened and organized, perhaps, but what one would gain from such a process is not entirely clear.

23.1.1 Reduction of an NHL Game

For a meatier example, consider the NHL game that took place between the Ottawa Senators and the Toronto Maple Leafs on February 18, 2017 [21].

As a first approximation, we shall think of a hockey game as a series of sequential and non-overlapping "events" involving two teams of skaters. What does it mean to have extracted useful insights from such a series of events?

At some level, the most complete raw understanding of that night's game belongs to the game's active and passive participants (players, referees, coaches, general managers, official scorer and time-keeper, etc.).[1]

1: This simple assumption is rather old-fashioned and would be disputed by many in the age of hockey analytics, but we let it stand for now.

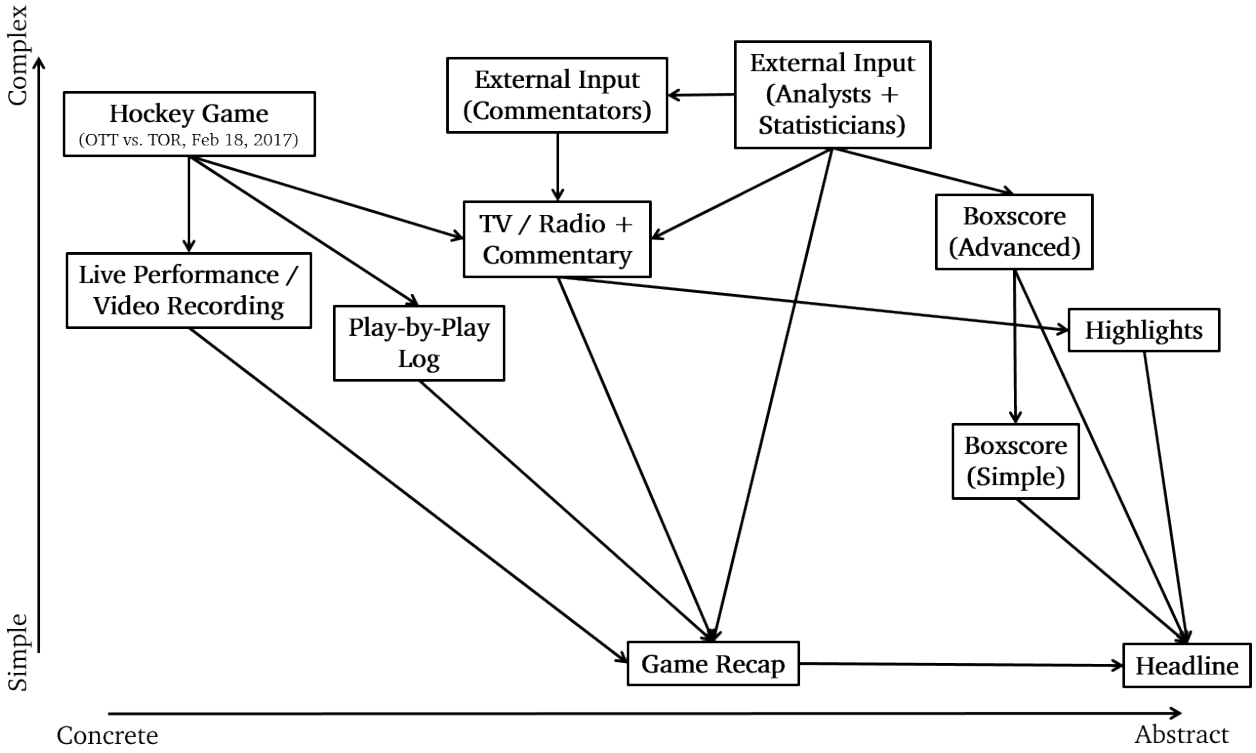


Figure 23.1: A schematic diagram of data reduction as it could apply to a professional hockey game.

The larger group of individuals who attended the game in person, watched it on TV/Internet, or listened to it on the radio presumably also have a lot of the facts at their disposal, with some contamination, as it were, by commentators (in the two latter cases).

Presumably, the participants and the witnesses also possess insights into the specific game: how could that information best be relayed to members of the public who did not catch the game? There are many ways to do so, depending on the intended level of abstraction and on the target audience (see Figure 23.1).

Play-by-Play Text File If a hockey game is a series of events, why not simply list the events, in the order in which they occurred? Of course, not everything that happens in the "**raw**" game requires reporting – it might be impressive to see Auston Matthews skate by Dion Phaneuf on his way to the Senators' net at the 8:45 mark of the 2nd period, say, but reporting this "event" would only serve to highlight the fact that Matthews is a better skater than Phaneuf. It is true, to be sure, but some level of filtering must be applied in order to retain only **relevant** (or "high-level") information, such as:

> blocked shots, face-off wins, giveaways, goals, hits, missed shots, penalties, power play events, saves, shorthanded events, shots on goal, stoppage (goalie stopped, icing, offside, puck in benches), takeaways, etc.

In a typical game, between 300 and 400 events are recorded (see Figure 23.2 for an extract of the play-by-play file for the game under consideration; the full list is found at [21]).

17:41	Toronto	Ben Smith won faceoff in defensive zone
17:46	**Ottawa**	**Goal scored by Ryan Dzingel assisted by Marc Methot and Mark Stone**
17:46	Ottawa	Kyle Turris won faceoff in neutral zone
18:14		Stoppage - Puck in Netting
18:14	Ottawa	Penalty to Bobby Ryan 2 minutes for Delaying the game
18:14	Ottawa	Shorthanded - Zack Smith won faceoff in neutral zone
18:42	Toronto	Power play - James van Riemsdyk shot blocked by Cody Ceci
18:42		Stoppage - Puck in Netting
18:42	Toronto	Power play - Auston Matthews won faceoff in offensive zone
18:48	Toronto	Power play - Giveaway by Jake Gardiner in offensive zone
19:15	Toronto	Power play - Connor Brown credited with hit on Jean-Gabriel Pageau in offensive zone
19:24	Toronto	Power play - Shot missed by William Nylander
20:00		End of 1st period

2nd Period Summary

Time	Team	Detail
0:00		Start of 2nd period
0:00	Toronto	Power play - Auston Matthews won faceoff in neutral zone
0:35	Ottawa	Takeaway by Marc Methot in defensive zone
1:10	Toronto	Josh Leivo credited with hit on Cody Ceci in offensive zone
1:16	Ottawa	Giveaway by Dion Phaneuf in defensive zone

Figure 23.2: Play-by-play extract, Ottawa Senators Toronto Maple Leafs, February 18, 2017 [21].

Some knowledge about the sport is required to make sense of some of the entries (colouring, use of bold text, etc.), but with patience we can rather easily[2] re-constitute the flow of the game.

2: That is to say, mechanically.

This approach, as we can see, is **fully descriptive**.

Boxscore The play-by-play does convey the game's events, but the relevance of its entries is sometimes questionable. In the general context of the game, how useful is to know that Nikita Zaitsev blocked a shot by Erik Karlsson at the 2:38 mark of the 1st period? Had this blocked shot saved a guaranteed Senators goal or directly lead to a Maple Leafs goal, one could have argued for its inclusion in the list of crucial events to report, but only the most fastidious observer[3] would bemoan its removal from the game's report.

3: Or a statistical analyst

The game's boxscore provides relevant information, at the cost of **completeness**: it distills the play-by-play file into a series of meaningful **statistics** and **summaries**, providing insights into the game that even a fan in attendance might have missed while the game was going on (see Figures 23.3-23.5).

Once again, a certain amount of **knowledge** about the sport is required to make sense of the statistics – to place them in the right context: is it meaningful that the Senators won 36 faceoffs to the Maple Leafs' 31? That Mark Stone was a +4 on the night? That both teams went "1-for-4' ' on the powerplay?

We cannot re-constitute the full flow of the game from the boxscore alone, but the approach is not solely descriptive – questions can be asked, and answers provided... the **analytical** game is afoot!

Game Information

Arena: Air Canada Centre
Location: Toronto, Ontario
Referees: Brian Pochmara, Brad Watson
Linesmen: Bevan Mills, Scott Driscoll
Attendance: 19,527 (103.9% full)

Team Statistical Comparison

Key: Ottawa Toronto

Total Shots	PIM	Hits	Giveaways	Takeaways	Faceoffs Won
▶ 42	10	▶ 32	▶ 7	9	▶ 36
37	10	23	6	▶ 12	31

1st Period Summary

Time	Team	Scoring Detail	OTT	TOR
17:26		Chris Wideman (4) *Assists: Derick Brassard, Mark Stone*	1	0
17:46		Ryan Dzingel (12) *Assists: Marc Methot, Mark Stone*	2	0

Time	Team	Penalty Detail
3:12		Roman Polak: 2 Minutes for Roughing
3:12		Mark Borowiecki: 2 Minutes for Roughing
8:03		Jake Gardiner: 2 Minutes for Slashing
12:21		Tom Pyatt: 2 Minutes for Hooking
13:59		Kyle Turris: 2 Minutes for Holding
18:14		Bobby Ryan: 2 Minutes for Delaying Game - Puck over Glass

2nd Period Summary

Time	Team	Scoring Detail	OTT	TOR
14:38		Morgan Rielly (3) *Assists: Auston Matthews, William Nylander*	2	1
17:52		Nazem Kadri (24) *Assist: Josh Leivo*	2	2

Time	Team	Penalty Detail
9:29		Matt Martin: 2 Minutes for Interference
9:40		Zach Hyman: 2 Minutes for Holding

3rd Period Summary

Time	Team	Scoring Detail	OTT	TOR
2:04		William Nylander (16) (Power Play) *Assists: Auston Matthews, Leo Komarov*	2	3
5:32		Mike Hoffman (19) *Assists: Kyle Turris, Erik Karlsson*	3	3
6:26		Derick Brassard (10) (Power Play) *Assists: Mark Stone, Erik Karlsson*	4	3
18:10		Mark Stone (21) *Assist: Kyle Turris*	5	3
19:15		Derick Brassard (11) *Assists: Kyle Turris, Mark Stone*	6	3

Time	Team	Penalty Detail
0:45		Zack Smith: 2 Minutes for Hooking
6:12		Nazem Kadri: 2 Minutes for Holding

Figure 23.3: Advanced Boxscore (I), Ottawa Senators Toronto Maple Leafs, February 18, 2017 [21].

Player Summary

Ottawa Senators

													Time On Ice			Faceoffs		
Player	**G**	**A**	**+/-**	**SOG**	**MS**	**BS**	**PN**	**PIM**	**HT**	**TK**	**GV**	**SHF**	**TOT**	**PP**	**SH**	**EV**	**FW**	**FL**	**%**
M. Borowiecki D	0	0	1	1	1	2	1	2	8	0	0	23	12:30	0:00	0:36	11:54	0	0	00.0
D. Brassard C	2	1	3	2	2	0	0	0	1	0	0	28	15:43	1:52	0:00	13:51	4	11	26.7
C. Ceci D	0	0	0	5	0	3	0	0	2	0	0	32	23:06	0:00	4:13	18:53	0	0	00.0
R. Dzingel C	1	0	1	3	1	0	0	0	0	1	1	19	11:56	1:10	0:00	10:46	0	0	00.0
M. Hoffman LW	1	0	1	6	3	0	0	0	0	1	0	23	15:34	2:55	0:00	12:39	0	0	00.0
E. Karlsson D	0	2	2	3	1	1	0	0	1	0	2	33	24:57	3:49	2:44	18:24	0	0	00.0
C. Kelly C	0	0	0	3	0	0	0	0	3	1	0	18	12:37	0:00	2:13	10:24	0	3	00.0
M. Methot D	0	1	2	1	0	1	0	0	4	2	0	32	21:12	0:00	2:52	18:20	0	0	00.0
C. Neil RW	0	0	0	0	0	0	0	0	1	0	0	6	3:55	0:00	0:00	3:55	0	0	00.0
J. Pageau C	0	0	0	1	0	3	0	0	3	0	0	30	18:17	0:36	3:57	13:44	13	6	68.4
D. Phaneuf D	0	0	0	4	0	1	0	0	2	0	1	35	24:13	2:33	3:29	18:11	0	0	00.0
T. Pyatt C	0	0	0	0	1	1	1	2	1	1	1	27	15:31	0:16	2:42	12:33	0	1	00.0
B. Ryan RW	0	0	-1	1	0	3	1	2	1	0	0	21	12:11	0:57	0:00	11:14	0	0	00.0
Z. Smith C	0	0	-1	1	2	0	1	2	3	1	1	23	16:26	0:31	2:05	13:50	3	5	37.5
M. Stone RW	1	4	4	5	0	1	0	0	2	2	0	26	18:38	3:12	0:00	15:26	0	0	00.0
K. Turris C	0	3	2	3	0	2	1	2	0	0	0	31	21:56	3:18	0:31	18:07	15	5	75.0
C. Wideman D	1	0	1	3	1	0	0	0	0	0	0	19	12:35	0:56	0:00	11:39	0	0	00.0
T. Wingels C	0	0	0	0	0	0	0	0	0	0	0	16	9:24	0:00	2:04	7:20	1	0	100.0

Ottawa Senators Scratches

Player	Detail
C. Lazar	Scratched
F. Claesson	Scratched

Toronto Maple Leafs

													Time On Ice			Faceoffs		
Player	**G**	**A**	**+/-**	**SOG**	**MS**	**BS**	**PN**	**PIM**	**HT**	**TK**	**GV**	**SHF**	**TOT**	**PP**	**SH**	**EV**	**FW**	**FL**	**%**
T. Bozak C	0	0	-3	3	0	0	0	0	0	2	0	23	16:41	3:04	0:00	13:37	6	9	40.0
C. Brown RW	0	0	-3	1	1	2	0	0	3	1	0	24	18:00	3:23	0:52	13:45	0	0	00.0
C. Carrick D	0	0	-1	0	0	0	0	0	0	0	0	24	18:01	0:30	0:00	17:31	0	0	00.0
J. Gardiner D	0	0	-1	1	1	1	1	2	0	0	2	28	21:49	3:31	0:00	18:18	0	0	00.0
M. Hunwick D	0	0	0	0	0	2	0	0	0	0	0	22	15:11	0:00	2:34	12:37	0	0	00.0
Z. Hyman C	0	0	0	2	2	0	1	2	0	1	0	24	16:11	0:14	1:33	14:24	0	1	00.0
N. Kadri C	1	0	-1	5	0	0	1	2	0	1	1	27	16:13	3:20	0:00	12:53	9	5	64.3
L. Komarov C	0	1	0	3	2	0	0	0	3	0	0	27	18:24	3:41	2:38	12:05	1	3	25.0
J. Leivo LW	0	1	0	1	1	1	0	0	1	3	0	24	14:38	3:04	0:00	11:34	0	0	00.0
M. Martin LW	0	0	-1	0	0	0	1	2	2	1	0	12	7:56	0:00	0:11	7:45	0	1	00.0
A. Matthews C	0	2	0	4	2	0	0	0	0	1	2	27	18:54	3:57	0:00	14:57	11	11	50.0
W. Nylander RW	1	1	0	2	3	0	0	0	0	0	0	26	18:54	3:53	0:00	15:01	0	0	00.0
R. Polak D	0	0	0	2	0	0	1	2	3	0	1	23	14:42	0:00	2:34	12:08	0	0	00.0
M. Rielly D	1	0	-2	5	2	2	0	0	2	0	0	35	21:29	1:12	1:51	18:26	0	0	00.0
B. Smith RW	0	0	-1	0	0	0	0	0	1	0	0	15	9:01	0:00	0:51	8:10	4	6	40.0
N. Soshnikov RW	0	0	0	2	0	0	0	0	1	1	0	14	9:14	0:00	0:56	8:18	0	0	00.0
J. van Riemsdyk LW	0	0	-2	3	0	1	0	0	2	0	0	23	14:52	2:46	0:00	12:06	0	0	00.0
N. Zaitsev D	0	0	-2	2	2	2	0	0	5	1	0	37	22:45	2:14	1:51	18:40	0	0	00.0

Toronto Maple Leafs Scratches

Player	Detail
A. Marchenko	Scratched
M. Marner	Upper Body
M. Marincin	Scratched

Figure 23.4: Advanced Boxscore (II), Ottawa Senators Toronto Maple Leafs, February 18, 2017 [21].

Goaltending Summary

Ottawa Senators Goaltending

Player	SA	GA	Saves	SV%	TOI	PIM
C. Anderson	37	3	34	.919	60:00	0

Toronto Maple Leafs Goaltending

Player	SA	GA	Saves	SV%	TOI	PIM
F. Andersen	40	4	36	.900	58:51	0

Shots On Goal

Team	1st	2nd	3rd	T
Ottawa	14	16	12	42
Toronto	15	10	12	37

Power Play Summary

Team	PPG / PPO
Ottawa	1 of 4
Toronto	1 of 4

Figure 23.5: Advanced Boxscore (III), Ottawa Senators Toronto Maple Leafs, February 18, 2017 [21].

Recap/Highlights One of the boxscore's shortcomings is that it does not provide much in the way of **narrative**, which has become a staple of sports reporting – what really happened during that game? How does it impact the current season for either team?

> **Associated Press, 19 February 2017 TORONTO**
> *The Ottawa Senators have the Atlantic Division lead in their sights.*
>
> Mark Stone had a goal and four assists, Derick Brassard scored twice in the third period and the Senators recovered after blowing a two-goal lead to beat the Toronto Maple Leafs 6-3 on Saturday night.
>
> The Senators pulled within two points of Montreal for first place in the Atlantic Division with three games in hand. "We like where we're at. We're in a good spot," Stone said. "But there's a little bit more that we want. Obviously, there's teams coming and we want to try and create separation, so the only way to do that is keep winning hockey games."
>
> Ottawa led 2-0 after one period but trailed 3-2 in the third before getting a tying goal from Mike Hoffman and a power-play goal from Brassard. Stone and Brassard added empty-netters, and Chris Wideman and Ryan Dzingel also scored for the Senators. Ottawa has won four of five overall and three of four against the Leafs this season. Craig Anderson stopped 34 shots.
>
> Morgan Rielly, Nazem Kadri and William Nylander scored and Auston Matthews had two assists for the Maple Leafs. Frederik Andersen allowed four goals on 40 shots. Toronto has lost eight of 11 and entered the night with a tenuous grip on the final wild-card spot in the Eastern Conference.
>
> "The reality is we're all big boys, we can read the standings. You've got to win hockey games," Babcock said. After Nylander made it 3-2 with a power-play goal 2:04 into the third, Hoffman tied it by rifling a shot from the right faceoff circle off the post and in. On a power play 54 seconds later, Andersen stopped Erik Karlsson's point shot, but Brassard jumped on the rebound and put it in for a 4-3 lead.
>
> Wideman started the scoring in the first, firing a point shot through traffic moments after Stone beat Nikita Zaitsev for a puck behind

the Leafs goal. Dzingel added to the lead when he deflected Marc Methot's point shot 20 seconds later.

Andersen stopped three shots during a lengthy 5-on-3 during the second period, and the Leafs got on the board about three minutes later. Rielly scored with 5:22 left in the second by chasing down a wide shot from Matthews, carrying it to the point and shooting through a crowd in front.

About three minutes later, Zaitsev fired a shot from the right point that sneaked through Anderson's pads and slid behind the net. Kadri chased it down and banked it off Dzingel's helmet and in for his 24th goal of the season. Dzingel had fallen in the crease trying to prevent Kadri from stuffing the rebound in.

"Our game plan didn't change for the third period, and that's just the maturity we're gaining over time," Senators coach Guy Boucher said. "Our leaders have been doing a great job, but collectively, the team has grown dramatically in terms of having poise, executing under pressure."

Game notes: Mitch Marner sat out for Toronto with an upper-body injury. Marner leads Toronto with 48 points and is also expected to sit Sunday night against Carolina.

UP NEXT Senators: Host Winnipeg on Sunday night. Maple Leafs: Travel to Carolina for a game Sunday night.

Simple Boxscore A gambler or a member of a hockey pool might be interested in the fact that Auston Matthews spent nearly 4 minutes on the powerplay (see Figure 23.4), but a casual observer is likely to find the full boxscore a monstrous overkill. How much **crucial information** is lost/provided by the simple boxscore of Figure 23.6, instead?

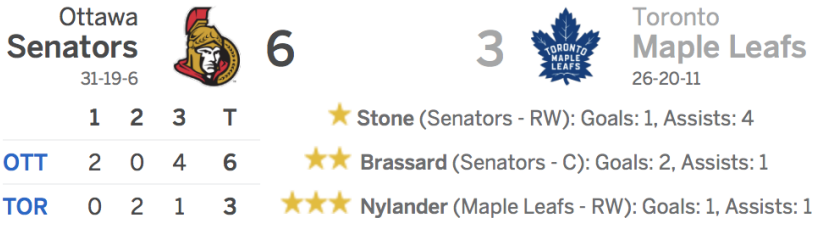


Figure 23.6: Simple boxscore, Ottawa Senators Toronto Maple Leafs, February 18, 2017 [21].

Headline If we take the view that humans **impose** a narrative on sporting events (rather than unearth it), we could argue that the only **"true" informational content** is found in the following headline:

> **Sens rally after blowing lead, beat Leafs, gain on Habs.** [21]

Visualization It is easy to get lost in row after row of statistics and events description, or in large bodies of text.[4] Visualizations can help complement our understanding of any data analytic situation.

While they can be appealing on their own, a certain amount of external context is required to make sense of most of them (see Figure 23.7).

4: Doubly so for a machine in the latter case.

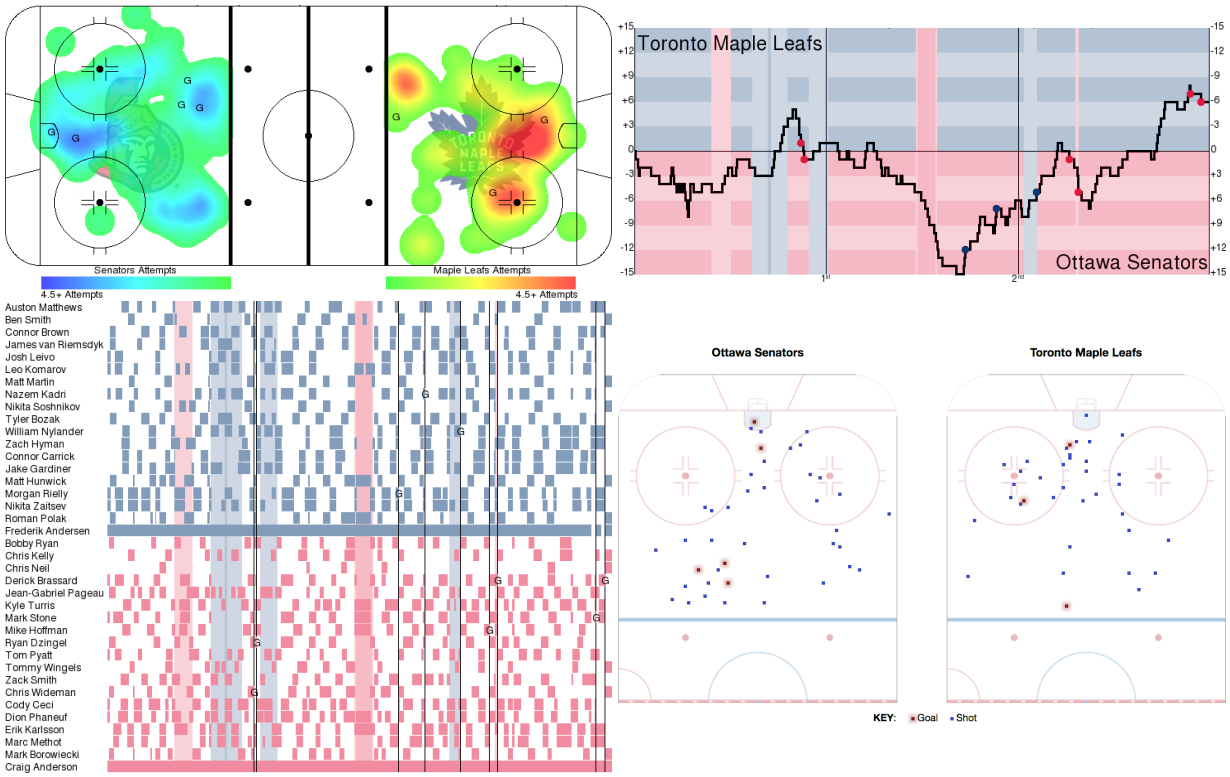


Figure 23.7: Visualizations, Ottawa Senators Toronto Maple Leafs, February 18, 2017: offensive zone unblocked shots heat map (top left), gameflow chart, Corsi +/- , all situations (top right), player shift chart (bottom left), shots and goals (bottom right) [17].

General Context A document which is prepared for analysis is often part of a **more general context** (or a collection).

Can the analysis of all the games between the Senators and the Maple Leafs shed some light on their rivalry on the ice? Obviously, the more arcane the representation method, the more in-depth knowledge of the game and its statistics is required, but to those in the know, **summaries** and **visualizations** can provide valuable insight (see Figure 23.8).

There are thus various ways to understand a single hockey game – and a series of games – depending on the desired (or required) levels of abstraction and complexity.

But as is the case for **all** quantitative methods, data reduction for insight is subject to **analytic choices** – you may have noticed that we conspicuously averted reporting on playoff results, and on post-2017 results. Would the overall "understanding" of the game in question (and the rivalry, in general) change if they were included?[5]

5: Unfortunately for this lifelong Sens fan, it most definitely would...

The specific details of data reduction as it applies to a professional hockey game are not usually **portable** to general situations, but the **main concepts** are, as we illustrate presently.

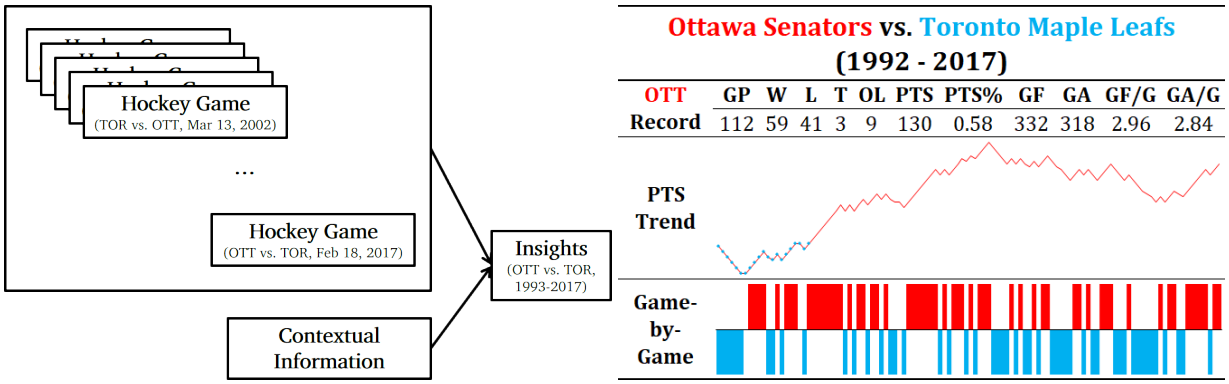


Figure 23.8: A schematic diagram of data reduction as it applies to a *corpus* of professional hockey games, with visualization and summarizing of regular season games between the Ottawa Senators and Toronto Maple Leafs (1993-2017).

23.1.2 Meaning in Macbeth

> It is a tale told by an idiot, full of sound and fury, signifying nothing. [*Macbeth*, V.5, line 30]

In a sense, in order to extract the full **meaning** out of a document, said document needs to be read and understood in its entirety.[6] But even if we have the luxury of doing so, some issues appear:

- do all readers extract the same meaning?
- does meaning stay constant over time?
- is meaning retained by the language of the document?
- do the author's intentions constitute the true (baseline) meaning?
- does re-reading the document change its meaning?

Given the uncertain nature of what a document's meaning actually is, it is counter-productive to talk about insight or meaning (in the **singular**); rather we look for insights and meanings (in the **plural**).

Consider the following passage from *Macbeth* (Act I, Scene 5, Lines 45-52):

> [Enter **MACBETH**]
> **LADY MACBETH**: Great Glamis, worthy Cawdor,
> Greater than both, by the all-hail hereafter,
> Thy letters have transported me beyond
> This ignorant present, and I feel now
> The future in the instant
> **MACBETH**: My dearest love, Duncan comes here tonight.
> **LADY MACBETH**: And when goes hence?
> **MACBETH**: Tomorrow, as he purposes.

What is the "meaning" of this scene? What is the "meaning" of *Macbeth* as a whole? As a starting point, it's crucial to note that the "meaning" of the scene is likely not independent of the play's context up to this scene.[7]

Does the plot description carry the same "meaning" as the play itself? What about TVTropes 🔗 's laconic description of *Macbeth* [25]:

> Hen-pecked Scottish nobleman murders his king and spends the rest of the play regretting it.

6: This section also serves as an introduction to Chapter 27 (*Text Analysis and Text Mining*).

7: A description of the plot in modern prose is provided in [27].

Or Mister Apple's haiku description (same site)?

> Macbeth and his wife
> Want to become the royals
> So they kill 'em all.

Or this literary description, from an unknown author?

> Macbeth dramatizes the battle between good and evil, exploring the psychological effects of King Duncan's murder on Macbeth and Lady Macbeth. His conflicting feelings of guilt and ambition embody this timeless battle of good vs evil.

Or yet again the (fantastic) 2001 movie *Scotland, PA*, featuring James LeGros, Maura Tierney, and Christopher Walken [16]?

For non-native English speakers (and for a number of native speakers as well, it should be said...), the play (to say nothing of the quoted passage above) might prove difficult to parse and understand.

A modern translation (which is a form of data reduction) is available at *No Fear Shakespeare*, shedding some light on the semantic role of the scene:

> ***MACBETH*** *enters.*
> **LADY MACBETH**: Great thane of Glamis! Worthy thane of Cawdor! You'll soon be greater than both those titles, once you become king! Your letter has transported me from the present moment, when who knows what will happen, and has made me feel like the future is already here.
> **MACBETH**: My dearest love, Duncan is coming here tonight.
> **LADY MACBETH**: And when is he leaving?
> **MACBETH**: He plans to leave tomorrow.

Consider, also, the French translation by F. Victor Hugo:

> *Entre* ***MACBETH****.*
> **LADY MACBETH**, *continuant*: Grand Glamis! Digne Cawdor! plus grand que tout cela par le salut futur! Ta lettre m'a transportée au delà de ce présent ignorant, et je ne ne sens plus dans l'instant que l'avenir.
> **MACBETH**: Mon cher amour, Duncan arrive ici ce soir.
> **LADY MACBETH**: Et quand repart-il?
> **MACBETH**: Demain... C'est son intention.

Do these all carry the same *Macbeth* essence? Do they all even carry a *Macbeth* essence? Are they all *Macbeth*? How much, if anything, of *Macbeth* do they preserve? The French translation, for instance, adds a very ominous tone to Macbeth's last retort to his wife.

Those of us who have read the rest of the play know that the tone is in keeping with the events that will eventually transpire, but does the translation add some foreshadowing that is simply not present up to that point in the original? If so, does it matter?

23.2 Dimension Reduction

There are advantages to working with **reduced, low-dimensional data**:

- **visualisation methods** of all kinds are available and (more) readily applicable to such data in order to extract and present insights;
- high-dimensional data is subject to the **curse of dimensionality** (CoD), which asserts (among other things) that multi-dimensional spaces are ... well, **vast**, and when the number of features in a model **increases**, the number of observations required to maintain predictive power also **increases**, but at a **substantially larger rate**;
- a consequence of CoD is that in high-dimension sets, all observations are roughly **dissimilar** to one another – observations tend to be **nearer** the dataset's **boundaries** than to one another.

Dimension reduction techniques such as the ubiquitous **principal component analysis**, **independent component analysis**, **factor analysis**,[8] or **multiple correspondence analysis**[9] project multi-dimensional datasets onto **low-dimensional** but **high-information** spaces.[10]

Some information is necessarily lost in the process, but in many instances the drain can be kept under control and the gains made by working with smaller datasets can offset the loss of completeness.

8: For numerical data.

9: For categorical data.

10: The so-called **Manifold Hypothesis**, see Section 23.2.4.

23.2.1 Sampling Observations

Datasets can be "**big**" in a variety of ways:

- they can be too large for the **hardware** to handle,[11] or
- the dimensions can go against specific *modeling assumptions*.[12]

For instance, multiple sensors which record 100+ observations per second in a large geographical area over a long time period can lead to excessively big datasets, say.

A natural question, regarding such a dataset, is whether every one of its row needs to be used: if rows are selected randomly (with or without replacement), the resulting sample might be **representative**[13] of the entire dataset, and the smaller set might be easier to handle.

There are some drawbacks to the sampling approach, however:

- if the signal of interest is **rare**, sampling might lose its presence altogether;
- if **aggregation** happens at some point in the reporting process, sampling will necessarily affect the totals,[14] and
- even **simple** operations on large files (finding the # of rows, say) can be taxing on the memory or computation time – some knowledge or **prior information** about the dataset structure can help.

Sampled datasets can also be used to work the kinks out of the data analysis workflows, but the key take-away is that if data is too big to store, access, and manipulate in a reasonable amount of time, the issue is mostly a **Big Data problem** – this is the time to start considering the use of distributed computing (see Chapter 30, *What's the Big Deal with Big Data?*).

11: That is to say, they cannot be stored or accessed properly due to the number of observations, the number of features, or the overall size of the dataset.

12: Such as the number of features being much larger than the number of observations, say.

13: An entire field of statistical endeavour – **statistical survey sampling** – has been developed to quantify the extent to which the sample is representative of the population, see Chapter 10 (*Survey Sampling Methods*).

14: For instance, if we are interested in predicting the number of passengers per flight leaving YOW (Macdonald-Cartier International Airport) and the total population of passengers is sampled, then the sampled number of passengers per flight is necessarily below the actual number of passengers per flight. Estimation methods exist to overcome these issues (see Chapter 10).

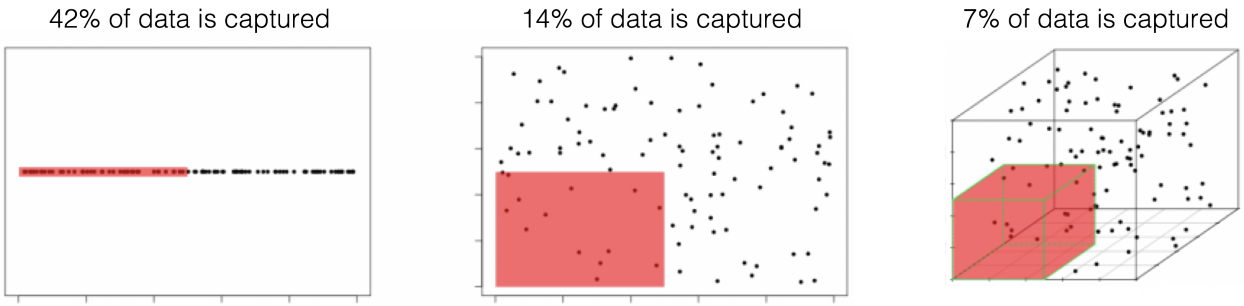


Figure 23.9: Illustration of the curse of dimensionality; $N = 100$ observations are uniformly distributed on the unit hypercube $[0,1]^d$, $d = 1,2,3$. The red regions represent the smaller hypercubes $[0,0.5]^d$, $d = 1,2,3$. The percentage of captured data points is seen to decrease with an increase in d [11].

23.2.2 The Curse of Dimensionality

A model is said to be **local** if it depends solely on the observations near the input vector (k nearest neigbours classification is local, whereas linear regression is global). With a large training set, increasing k in a kNN model, say, will yield enough data points to provide a solid approximation to the theoretical classification boundary.

The **curse of dimensionality** (CoD) is the breakdown of this approach in high-dimensional spaces: when the number of features increases, the number of observations required to maintain predictive power also increases, **but at a substantially higher rate** (see Figure 23.9 for an illustration of the CoD in action).

Manifestations of CoD Let $x_i \sim U^1(0,1)$ be i.i.d. for $i = 1,\dots,n$. For any $z \in [0,1]$ and $\varepsilon \in (0,1]$ such that

$$I_1(z;\varepsilon) = \{y \in \mathbb{R} : |z-y|_\infty < \varepsilon/2\} \subseteq [0,1],$$

the expected number of observations x_i in $I_1(z;\varepsilon)$ is

$$\left|I_1(z;\varepsilon) \cap \{x_i\}_{i=1}^n\right| \approx \varepsilon \cdot N,$$

so an ε_∞−ball subset of $[0,1]^1$ contains approximately ε of the observations in $\{x_i\}_{i=1}^n \subseteq \mathbb{R}$, **on average**.

Let $\mathbf{x}_i \sim U^2(0,1)$ be i.i.d. for all $i = 1,\dots,n$. For any $\mathbf{z} \in [0,1]^2$ and $\varepsilon \in (0,1]$ such that

$$I_2(\mathbf{z};\varepsilon) = \{\mathbf{Y} \in \mathbb{R}^2 : \|\mathbf{z}-\mathbf{Y}\|_\infty < \varepsilon/2\} \subseteq [0,1]^2,$$

the expected number of observations $\mathbf{x}_i$ in $I_2(\mathbf{z};\varepsilon)$ is

$$\left|I_1(\mathbf{z};\varepsilon) \cap \{\mathbf{x}_i\}_{i=1}^n\right| \approx \varepsilon^2 \cdot N,$$

so an ε_∞−ball subset of $[0,1]^2$ contains approximately ε^2 of the observations in $\{\mathbf{x}_i\}_{i=1}^n \subseteq \mathbb{R}^2$, **on average**.

In general, this reasoning shows that an ε_∞−ball subset of $[0,1]^p \subseteq \mathbb{R}^p$ contains approximately ε^p of the observations in $\{\mathbf{x}_i\}_{i=1}^n \subseteq \mathbb{R}^p$, **on average**.

To capture $r\%$ of uniformly distributed observations in a unit $p-$hypercube, we need a $p-$hypercube with edge $\varepsilon_p(r) = r^{1/p}$, on average. For instance, to capture $r = 1/3$ of the observations in a unit $p-$hypercube in $\mathbb{R}$, $\mathbb{R}^2$, and $\mathbb{R}^{10}$, a hyper-subset with edge $\varepsilon_1(1/3) \approx 0.33$, $\varepsilon_2(1/3) \approx 0.58$, and $\varepsilon_{10}(1/3) \approx 0.90$, respectively.

The inference is simple, but far-reaching: in general, as p increases, the nearest observations to a given point $\mathbf{x}_j \in \mathbb{R}^p$ are in fact quite distant from $\mathbf{x}_j$, in the Euclidean sense, on average – **locality is lost**![15]

15: The situation may not be as stark if the observations are not i.i.d., but the principle remains the same – in high-dimensional spaces, it is harder for observations to be near one another than it is so in low-dimensional spaces.

This can wreak havoc on models and algorithms that rely on the (Euclidean) nearness of observations (k nearest neighbours, $k-$means clustering, etc.). The CoD manifests itself in various ways.

In datasets with a large number of features:

- most observations are **nearer the edge of the sample than they are to other observations**, and
- realistic training sets are necessarily **sparse**.

Imposing restrictions on models can help mitigate the effects of the CoD, but if they are not warranted the end result may be catastrophic.

23.2.3 Principal Component Analysis

Principal component analysis (PCA) can be used to find the combinations of variables along which the data points are **most spread out**; it attempts to fit a $p-$**ellipsoid** to a centered representation of the data. The ellipsoid axes are the **principal components** of the data.

Small axes are components along which the variance is "small"; removing these components leads, in an ideal setting, to a "small" loss of information[16] (see Figure 23.10). The procedure is simple:

1. centre and "scale" the data to obtain a matrix $\mathbf{X}$;[17]
2. compute the data's covariance matrix $\mathbf{K} = \mathbf{X}^\top\mathbf{X}$;
3. find $\mathbf{K}$'s eigenvalues $\mathbf{\Lambda}$ and its orthonormal eigenvectors matrix $\mathbf{W}$;
4. each eigenvector $\mathbf{w}$ (also known as **loading**) represents an axis, whose variance is given by the associated eigenvalue λ.

16: Although there are scenarios where it could be those "small" axes that are more interesting – such as is the case with the "pancake stack" problem.

17: This is NOT the design matrix as defined in regression analysis.

The loading that explains the most variance along a single axis (the **1st PC**) is the eigenvector of the empirical covariance matrix corresponding to the largest eigenvalue, and that variance is proportional to the eigenvalue; the 2nd largest eigenvalue and its corresponding eigenvector yields the **2nd PC** and variance pair, and so on, yielding **orthonormal** principal components $\mathrm{PC}_1, \ldots, \mathrm{PC}_r$, where $r = \mathrm{rank}(\mathbf{X})$.[18]

18: If some of the eigenvalues are 0, then $r < p$, and *vice-versa*, implying that the data was embedded in a $r-$dimensional manifold to begin with.

PCA can provide an avenue for dimension reduction, by "removing" components with small eigenvalues (as in Figure 23.10). The **proportion of the spread in the data** which can be explained by each principal component can be placed in a **scree plot** (a plot of eigenvalues against ordered component indices), and we retain the ordered PCs:

- for which the eigenvalue is above some threshold (say, 25%);
- for which the cumulative proportion of the spread falls below some threshold (say 95%), or
- prior to a **kink** in the scree plot.

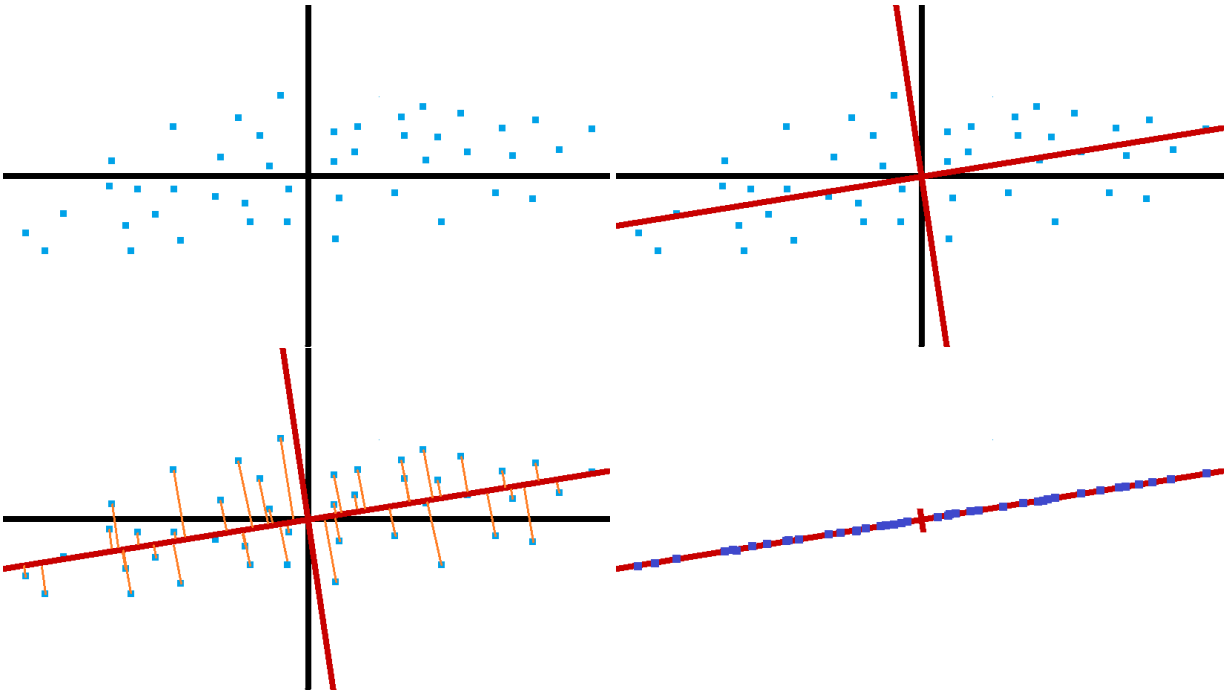


Figure 23.10: Illustration of PCA on an artificial 2D dataset (top left). The red axes (top right) represent the axes of the best elliptic fit. Removing the minor axis by projecting the points on the major axis (bottom left) leads to a dimension reduction and a (small) loss of information (bottom right).

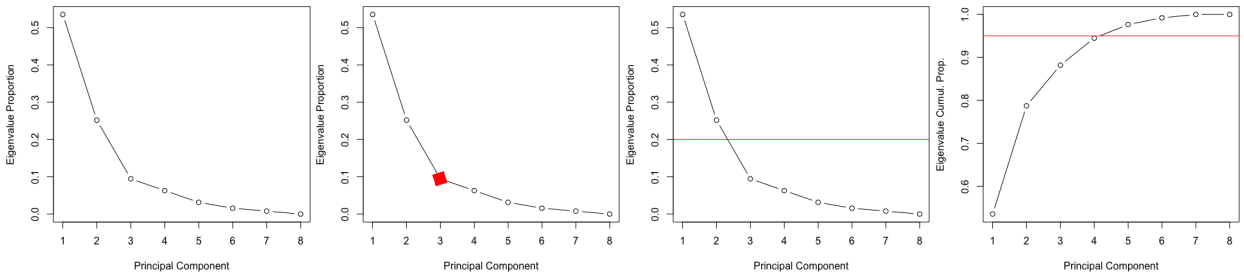


Figure 23.11: Selecting the number of principal component – the proportion of the variance explained by each (ordered) component is shown in the first 3 charts; the cumulative proportion is shown in the last chart. The kink method is shown in the top right image, the individual threshold component in the bottom left image, and the cumulative proportion in the bottom right image.

For instance, consider an 8−dimensional dataset for which the ordered PCA eigenvalues are provided below:

PC	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Var	17	8	3	2	1	0.5	0.25	0
Prop	54	25	9	6	3	2	1	0
Cumul	54	79	88	94	98	99	100	100

If the only PCs that are retained are those that explain up to 95% of the cumulative variation, say, then the original data reduces to a 4-dimensional subset; if only the PCs that individually explain more than 25% of the variation are retained, say, then the data reduces to a 2-dimensional subset; if only the PCs that lead into the first kink in the scree plot are retained, to a 3-dimensional subset (see Figure 23.11).

PCA is commonly-used, but often without regard to its inherent **limitations**, unfortunately:

- it is dependent on scaling, and so is not uniquely determined;
- with little domain expertise, it may be difficult to interpret the PCs;
- it is quite sensitive to outliers;
- the analysis goals are not always aligned with the principal components, and
- the data assumptions are not always met – in particular, does it always make sense that important data structures and data spread be correlated (the so-called **counting pancakes** problem), or that the components be **orthogonal**?

There are other methods to find the **principal manifolds** of a dataset, including UMAP, self-organizing maps, auto-encoders, curvilinear component analysis, manifold sculpting, kernel PCA, etc.

Formalism Because $\mathbf{K}$ is **positive semi-definite** ($\mathbf{K} \geq 0$), the eigenvalues $\lambda_i = s_i^2$ are non-negative and they can be ordered in a decreasing sequence

$$\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_p), \quad \text{where } \lambda_1 \geq \cdots \lambda_p \geq 0$$

and $\mathbf{W} = [\mathbf{w}_1 | \cdots | \mathbf{w}_p]$.

If $k = \text{rank}(\mathbf{X})$, then there are $p - k$ "empty" principal component (corresponding to null eigenvalues) and k "regular" principal components (corresponding to zero eigenvalues). We write $\mathbf{W}^* = [\mathbf{w}_1 | \cdots | \mathbf{w}_k]$ and $\mathbf{\Lambda}^* = \text{diag}(\lambda_1, \ldots, \lambda_k)$. If $p - k \neq 0$, then the eigenvalue decomposition of $\mathbf{K}$ is

$$\mathbf{K} = \begin{bmatrix} \mathbf{W}^* & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{\Lambda}^* & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} (\mathbf{W}^*)^\top \\ \mathbf{0} \end{bmatrix} = \mathbf{W}\mathbf{\Lambda}\mathbf{W}^\top;$$

if $\mathbf{X}$ is of full rank, then $\mathbf{W}^* = \mathbf{W}$ and $\mathbf{\Lambda}^* = \mathbf{\Lambda}$.

The eigenvectors of K (the $\mathbf{w}_j$) are the **singular vectors** of $\mathbf{X}$: there exist $\mathbf{U}_{n \times n}$ and $\mathbf{\Sigma}_{n \times p}$ such that

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{W}^\top,$$

where

$$\mathbf{U} = \begin{pmatrix} \mathbf{U}^* & \mathbf{0} \end{pmatrix} \quad \text{and} \quad \mathbf{\Sigma} = \begin{pmatrix} \text{diag}(s_i) \\ \mathbf{0} \end{pmatrix}.$$

If $\mathbf{X}$ is of full rank, then $\mathbf{W}$ is **orthonormal** and so represents a **rotation matrix**. As $\mathbf{W}^{-1} = \mathbf{W}^\top$, we must then have $\mathbf{X}\mathbf{W} = \mathbf{U}\mathbf{\Sigma}$, the **principal component decomposition of X**:

$$\mathbf{T}_{n \times p} = \mathbf{X}\mathbf{W}, \quad \begin{bmatrix} \mathbf{t}_1 & \cdots & \mathbf{t}_p \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_n \end{bmatrix}^\top \begin{bmatrix} \mathbf{w}_1 & \cdots & \mathbf{w}_p \end{bmatrix}.$$

The link between the principal components and the eigenvectors can be made explicit: the first principal component PC_1 is the loading $\mathbf{w}_1$ (with $\|\mathbf{w}_1\|_2 = 1$) which **maximizes the variance of the first column** of $\mathbf{T}$:

$$\mathbf{w}_1 = \arg \max_{\|\mathbf{w}\|_2 = 1} \{\text{Var}(\mathbf{t}_1)\} = \arg \max_{\|\mathbf{w}\|_2 = 1} \left\{ \frac{1}{n-1} \sum_{i=1}^{n} (t_{1,i} - \overline{t_1})^2 \right\}.$$

Since

$$\overline{t_1} = \frac{1}{n}\sum_{j=1}^{n} \mathrm{E}[\mathbf{x}_j^\top \mathbf{w}_1] = \frac{1}{n}\sum_{j=1}^{n} \mathrm{E}\left[\sum_{i=1}^{p} x_{j,i} w_{i,1}\right]$$
$$= \frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{p} w_{i,1} \underbrace{\mathrm{E}[x_{j,i}]}_{=0} = 0,$$

then $\mathrm{Var}(\mathbf{t}_1) = \frac{1}{n-1}(t_{1,1}^2 + \cdots + t_{n,1}^2)$ and the problem is equivalent to

$$\mathbf{w}_1 = \arg\max_{\|\mathbf{w}\|_2=1} \{t_{1,1}^2 + \cdots + t_{n,1}^2\},$$

By construction, $t_{i,1}^2 = (\mathbf{x}_i^\top \mathbf{w}_1)^2$ for all i, so

$$t_{1,1}^2 + \cdots + t_{n,1}^2 = (\mathbf{x}_1^\top \mathbf{w}_1)^2 + \cdots + (\mathbf{x}_n^\top \mathbf{w}_1)^2 = \|\mathbf{X}\mathbf{w}_1\|_2 = \mathbf{w}_1 \mathbf{X}^\top \mathbf{X} \mathbf{w}_1.$$

Hence,

$$\mathbf{w}_1 = \arg\max_{\|\mathbf{w}\|_2=1} \{\mathbf{w}\mathbf{X}^\top \mathbf{X}\mathbf{w}\} = \arg\max_{\|\mathbf{w}\|_2=1} \{\mathbf{w}^\top \mathbf{K}\mathbf{w}\};$$

this is equivalent to finding the maximizer of $F(\mathbf{w}) = \mathbf{w}^\top \mathbf{K}\mathbf{w}$ subject to the constraint

$$G(\mathbf{w}) = 1 - \mathbf{w}^\top \mathbf{w} = 0.$$

We solve this problem by using the method of Lagarange multipliers; any optimizer $\mathbf{w}^*$ must be either:

1. a critical point of F, or
2. a solution of $\nabla F(\mathbf{w}) + \lambda \nabla G(\mathbf{w}) = \mathbf{0}$, $\lambda \neq 0$.

But $\nabla F(\mathbf{w}) = 2\mathbf{K}\mathbf{w}$ and $\nabla G(\mathbf{w}) = -2\mathbf{w}$; either $\mathbf{w}^* \in \ker(\mathbf{K})$ (case 1) or $2\mathbf{K}\mathbf{w}^* - 2\lambda^*\mathbf{w}^* = \mathbf{0}$ (case 2); either

$$\mathbf{K}\mathbf{w}^* = \mathbf{0} \quad \text{or} \quad (\mathbf{K} - \lambda^* I)\mathbf{w}^* = \mathbf{0},\ \lambda^* \neq 0.$$

In either case, $\lambda^* \geq 0$ is an eigenvalue of K, with associated eigenvector $\mathbf{w}^*$. There are at most p distinct possibilities $\{(\lambda_j, \mathbf{w}_j)\}_{j=1}^p$, and for each of them

$$\mathbf{w}_j^\top \mathbf{K}\mathbf{w}_j = \mathbf{w}_j^\top \lambda_j \mathbf{w}_j = \lambda_j \mathbf{w}_j^\top \mathbf{w}_j = \lambda_j,$$

since $\mathbf{w}_j^\top \mathbf{w}_j = 1$.

Thus,

$$\arg\max_{\|\mathbf{w}\|_2=1} \{\mathrm{Var}(\mathbf{t}_1)\} = \arg\max_{\|\mathbf{w}\|_2=1} \{\lambda_j\} = \mathbf{w}_1 = \mathrm{PC}_1,$$

since $\lambda_1 \geq \lambda \geq 0$ for all eigenvalues λ of $\mathbf{K}$.

A similar argument shows that $\mathbf{w}_j$, $j = 2, \ldots, p$, is the direction along which the variance is the jth highest, assuming that $\mathbf{w}_j$ is orthonormal to all the preceding $\mathbf{w}_\ell$, $\ell = 1, \ldots, j-1$, and that the variance is proportional to λ_j.

The process is repeated at most p times, yielding r non-trivial principal components $\mathrm{PC}_1, \ldots, \mathrm{PC}_r$, where $r \leq p$ is the rank($\mathbf{X}$). Thus, we see that the rotation matrix $\mathbf{W}$ that maximizes the variance sequentially in the columns of $\mathbf{T} = \mathbf{X}\mathbf{W}$ is the matrix of eigenvectors of $\mathbf{K} = \mathbf{X}^\top \mathbf{X}$.

We show how to implement principal component analysis in Sections 19.7.3 and 21.4.3 (in the Wine example).

23.2.4 The Manifold Hypothesis

Manifold learning involves mapping high-dimensional data to a lower dimensional manifold, such as mapping a set of points in $\mathbb{R}^3$ to a torus shape, which can then be **unfolded** (or embedded) into a 2D object.

Techniques for manifold learning are commonly-used because data is often (usually?) **sampled** from unknown and underlying sources which cannot be measured directly.

Learning a suitable "low-dimension" manifold from a higher-dimensional space is approximately as complicated as learning the sources (in a ML sense). This problem can also be re-cast as finding a set of **degrees of freedom** which can reproduce most of the variability in a dataset.

For instance, a set of multiple photographs of a 3D object taken from different positions but at the same distance from the object can be represented by two degrees of freedom: the **horizontal** and **vertical** angles from which the picture was taken.

As another example, consider a set of hand-written drawings of the digit "2" [24]. Each of these drawings can also be represented using a small number of degrees of freedom:

- the **ratio** of the length of the **lowest horizontal line** to the height of the hand-written **digit**;
- the **ratio** of the length of the **arch in the curve** at the top to the smallest horizontal distance from the **end point of the arch** to the **main vertical curve**;
- the **angle of rotation** of the digit as a whole with respect to some **baseline orientation**, etc.

These two scenarios are illustrated in Figure 23.12.

Dimensionality reduction and **manifold learning** are often used for one of three purposes:

- to **reduce the overall dimensionality** of the data while trying to **preserve the variance in the data**;
- to **display** high-dimensional datasets, or
- to **reduce the processing time** of supervised learning algorithms by lowering the dimensionality of the data.

PCA, for instance, provides a sequence of best linear approximations to high-dimensional observations (see previous section); the process has fantastic theoretical properties for computation and applications, but data is not always well-approximated by a fully linear process.

In this section, the focus is on non-linear dimensionality reduction methods, most of which are a variant of **kernel PCA**:

- LLE;
- Laplacian eigenmap;
- isomap;
- semidefinite embedding, and
- t–SNE.

By way of illustration, the different methods are applied to an "S"-shaped coloured 2D object living in 3D space (see Figure 23.15).

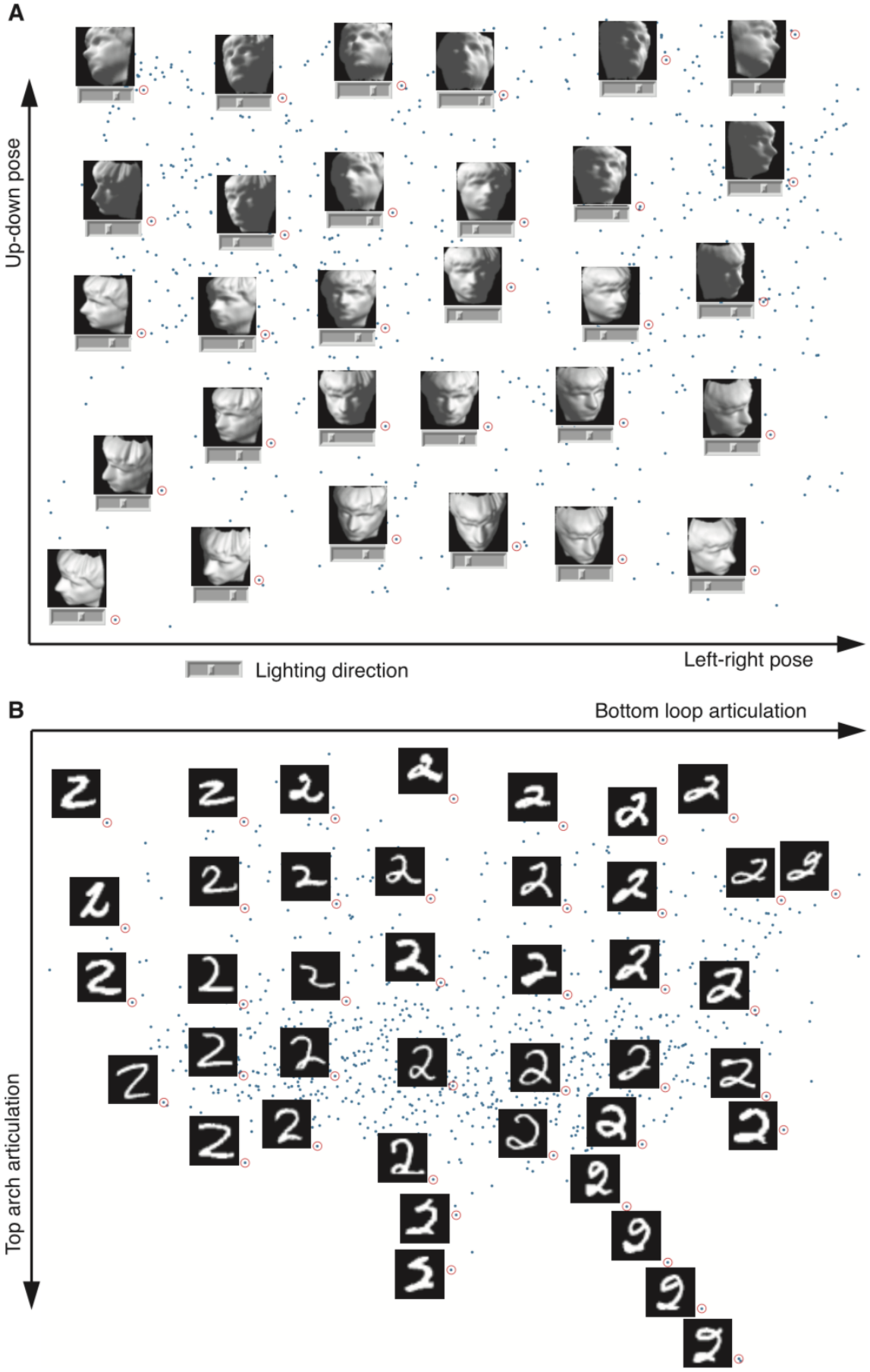


Figure 23.12: Plots showing degrees of freedom manifolds for images of faces (3D object) and handwritten digits [24].

Kernel Principal Component Analysis High-dimensional datasets often have a **non-linear nature**, in the sense that a linear PCA may only weakly capture/explain the variance across the entire dataset.

This is, in part, due to PCA relying on Euclidean distance as opposed to **geodesic distance** – the distance between two points *along* the manifold, that is to say, the distance that would be recorded if the high-dimensional object was first unrolled (see Figure 23.13).

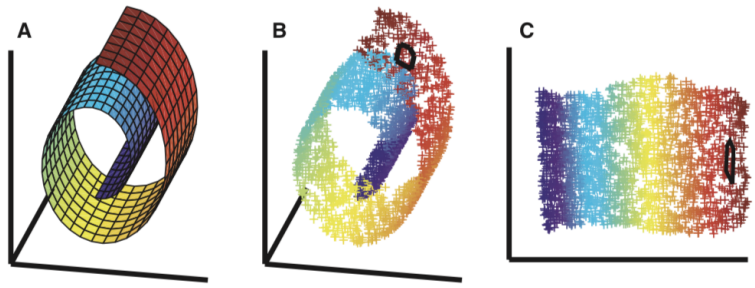


Figure 23.13: High-dimensional manifold unfolding; theoretical 2D manifold embedded in $\mathbb{R}^3$ (left), sample (middle), unfolding into $\mathbb{R}^2$ (right) [24].

Residents of Earth[19] are familiar with this concept: the Euclidean distance ("as the mole burrows") between Sao Paulo and Reykjavik is the length of the shortest tunnel joining the two cities, whereas the geodesic distance ("as the crow flies") is the arclength of the **great circle** through the two locations (see Figure 23.14).

19: Which we assume encompasses all of this work's readership...

Figure 23.14: Geodesic (red, solid) and Euclidean (orange, dash) paths between Sao Paulo and Reykjavik, Great Circle Map.

High-dimensional manifolds can be unfolded with the use of **transformations** Φ which map the input set of points

$$\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq \mathbb{R}^p$$

onto a **new set** of points

$$\{\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_n)\} \subseteq \mathbb{R}^m,$$

with $m \geq n$.

If Φ is chosen so that $\sum_{i=1}^{n} \Phi(\mathbf{x}_i) = \mathbf{0}$ (i.e., the transformed data is also centered in $\mathbb{R}^m$), we can formulate the kernel PCA objective in $\mathbb{R}^p$ as a linear PCA objective in $\mathbb{R}^m$:

$$\min \left\{ \sum_{i=1}^{n} \|\Phi(\mathbf{x}_i) - V_q V_q^\top \Phi(\mathbf{x}_i)\|^2 \right\},$$

over the set of $m \times q$ matrices V_q with orthonormal columns, where q is the desired dimension of the manifold.[20]

20: This **error reconstruction** approach to PCA yields the same results as the **covariance** approach of the previous section [8].

In practice, it can be difficult to determine Φ **explicitly**; in many instances, it is inner-product-like quantities that are of interest to the analyst.

The problem can be bypassed by working with **positive-definite kernel functions** $K : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}_+$ which satisfy $K(\mathbf{x}, \mathbf{y}) = K(\mathbf{y}, \mathbf{x})$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^p$ and

$$\sum_{i=1}^{k} \sum_{j=1}^{k} c_i c_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

for any integer k, coefficients $c_1, \ldots, c_k \in \mathbb{R}$ and vectors $\mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathbb{R}^p$, with equality if and only if $c_1, \cdots, c_k = 0$.[21]

21: These kernels also appear in **support vector machines** (see Section 21.4.2).

Popular data analysis kernels include:

- **linear kernel** $K(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{y}$;
- **ploynomial kernel** $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\top \mathbf{y} + r)^k$, $n \in \mathbb{N}$, $r \geq 0$, and
- **Gaussian kernel** $K(\mathbf{x}, \mathbf{y}) = \exp\left\{\frac{-\|\mathbf{x}-\mathbf{y}\|}{2\sigma^2}\right\}$, $\sigma > 0$.

Most dimension reduction algorithms can be re-expressed as some form of kernel PCA, as we will see shortly.

Locally Linear Embedding LLE is a manifold learning approach which addresses the problem of nonlinear dimension reduction by computing a low-dimensional, **neighbourhood-preserving embedding** of high-dimensional data.

The main assumption is that for any subset $\{\mathbf{x}_i\} \subseteq \mathbb{R}^p$ lying on some sufficiently well-behaved underlying $d-$dimensional manifold $\mathcal{M}$, each data point and its neighbours lie on a **locally linear patch** of $\mathcal{M}$. Using translations, rotations, and rescaling, the (high-dimensional) coordinates of each locally linear neighbourhood can be mapped to a set of $d-$dimensional global coordinates of $\mathcal{M}$.

This needs to be done in such a way that the relationships between neighbouring points are **preserved**. This can be achieved in 3 steps:

1. identify the **punctured neighbourhood** $N_i = \{i_1, \ldots, i_k\}$ of each data point $\mathbf{x}_i$ *via* k nearest neighbours;[22]
2. find the weights $z_{i,j}$ that provide the best linear reconstruction of each $\mathbf{x}_i \in \mathbb{R}^p$ from their respective punctured neighbourhoods[23] , i.e., solve

$$\min_{\mathbf{W}} \left\{ \sum_{i=1}^{n} \left\| \mathbf{x}_i - \sum_{j \in N_i} z_{i,j} \mathbf{x}_{N_i(j)} \right\|^2 \right\},$$

where $\mathbf{Z} = \left(z_{i,j}\right)$ is an $n \times n$ matrix ($z_{i,j} = 0$ if $j \notin N_i$), and

22: This could also be done by selecting all points within some fixed radius ε, but k is not a constant anymore, and that complicates matters.

23: Excluding $\mathbf{x}_i$ itself.

3. find the low-dimensional **embedding** vectors $\mathbf{y}_i \in \mathcal{M}(\subseteq \mathbb{R}^d)$ and neighbours $\mathbf{y}_{N_i(j)} \in \mathcal{M}$ for each i which are best reconstructed by the weights determined in the previous step, i.e., solve

$$\min_{\mathbf{Y}} \left\{ \sum_{i=1}^{n} \left\| \mathbf{y}_i - \sum_{j \in N_i} w_{i,j} \mathbf{y}_{N_i(j)} \right\|^2 \right\} = \min_{\mathbf{Y}} \left\{ \text{Tr}\left(\mathbf{Y}^\top \mathbf{Y} L\right) \right\},$$

where $L = (I - \mathbf{Z})^\top (I - \mathbf{Z})$ and $\mathbf{Y}$ is an $n \times d$ matrix.

If the global coordinates of the sampled points are **centered at the origin** and have **unit variance**,[24] it can be shown that L has a null eigenvalue with associated eigenvector.

[24] Which can always be achieved with an appropriate set of restrictions.

The j^{th} column of $\mathbf{Y}$ is the eigenvector associated with the j^{th} smallest non-zero eigenvalue of L [18].

Laplacian Eigenmap LE is similar to LLE, except that the first step consists in constructing a **weighted graph** $\mathcal{G}$ with n nodes (number of $p-$dimensional observations) and a set of edges connecting the neighbouring points.[25]

[25] As with LLE, the edges of $\mathcal{G}$ can be obtained by finding the k nearest neighbours of each node, or by selecting all points within some fixed radius ε.

In practice, the edges' **weights** are determined either:

- by using the inverse exponential with respect to the Euclidean distance
$$w_{i,j} = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{s}\right),$$
for all i, j, for some parameter $s > 0$, or
- by setting $w_{i,j} = 1$, for all i, j.

The embedding map is then provided by the following objective

$$\min_{\mathbf{Y}} \left\{ \sum_{i=1}^{n} \sum_{j=1}^{n} w_{i,j} (\mathbf{y}_i - \mathbf{y}_j)^2 \right\} = \min_{\mathbf{Y}} \left\{ \text{Tr}(\mathbf{Y} L \mathbf{Y}^\top) \right\},$$

subject to appropriate constraints, with the **Laplacian** $L = D - W$, where D is the (diagonal) **degree matrix** of $\mathcal{G}$,[26] and W its **weight matrix**.

[26] The sum of weights emanating from each node.

The Laplacian eigenmap construction is identical to the LLE construction, save for their definition of L.

Isomap This approach follows the same steps as LLE except that it uses **geodesic distance** instead of Euclidean distance when looking for each point's neighbours.[27]

[27] As always, neighbourhoods can be selected with kNN or with a fixed ε.

These neighbourhood relations are represented by a graph $\mathcal{G}$ in which each observation is connected to its neighbours *via* edges with weight $d_x(i, j)$ between neighbours.

The geodesic distances $d_{\mathcal{M}}(i, j)$ between all pairs of points on the manifold $\mathcal{M}$ are then estimated in the second step.

Semidefinite Embedding SDE requires learning $K(\mathbf{x}, \mathbf{z}) = \Phi(\mathbf{x})^\top \Phi(\mathbf{z})$ from the data before applying the kernel PCA transformation Φ, which is achieved by formulating the problem of learning K as an instance of **semidefinite programming**.

The distances and angles between observations and their neighbours are preserved under transformations by Φ: $\|\Phi(\mathbf{x_i}) - \Phi(\mathbf{x}_j)\|^2 = \|\mathbf{x}_i - \mathbf{x}_j\|^2$, for all $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^p$.

In terms of the kernel matrix, this constraint can be written as

$$K(\mathbf{x}_i, \mathbf{x}_i) - 2K(\mathbf{x}_i, \mathbf{x}_j) + K(\mathbf{x}_j, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|^2,$$

for all $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^p$.

By adding an objective function to maximize $\text{Tr}(K)$, that is, the variance of the observations in the learned feature space, SDE constructs a semidefinite program for learning the **kernel matrix**

$$K = \left(K_{i,j}\right)_{i,j=1}^n = \left(K(\mathbf{x}_i, \mathbf{x}_j)\right)_{i,j=1}^n,$$

from which we can proceed with kernel PCA.

Unified Framework All of the above algorithms (LLE, Laplacian Eigenmaps, Isomap, SDE) can be rewritten in the **kernel PCA** framework:

- in the case of LLE, if $\lambda_{\max}$ is the largest eigenvalue of

  $$L = (I - \mathbf{W})^\top (I - \mathbf{W}),$$

  then $K_{\text{LLE}} = \lambda_{\max} I - L$;
- with $L = D - W$, D a (diagonal) degree matrix with $D_{i,i} = \sum_{j=1}^n W_{i,j}$, then the corresponding K_{LE} is related to **commute times of diffusion** on the underlying graph, and
- with the Isomap element-wise squared geodesic distance matrix $\mathcal{D}^2$,

  $$K_{\text{Isomap}} = -\frac{1}{2}\left(I - \frac{1}{p}\mathbf{e}\mathbf{e}^\top\right)\mathcal{D}^2\left(I - \frac{1}{p}\mathbf{e}\mathbf{e}^\top\right),$$

  where $\mathbf{e}$ is a $p-$dimensional vector consisting solely of 1's (note that this kernel is not always positive semi-definite).

$t-$SNE There are a few relatively new manifold learning techniques that do not fit neatly in the kernel PCA framework: Uniform Manifold Approximation and Projection (UMAP, Section 23.4.4) and **$t-$Distributed Stochastic Neighbour Embedding** ($t-$SNE).

For a dataset $\{\mathbf{x}_i\}_{i=1}^n \subseteq \mathbb{R}^p$, the latter involves calculating probabilities

$$p_{i,j} = \frac{1}{2n}\left\{\frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)} + \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_j^2)}{\sum_{k\neq j}\exp(-\|\mathbf{x}_j - \mathbf{x}_k\|^2/2\sigma_j^2)}\right\},$$

which are proportional to the similarity of points in high-dimensional space $\mathbb{R}^p$ for all i, j, and $p_{i,i}$ is set to 0 for all i.[28] The bandwidths σ_i are selected in such a way that they are smaller in denser data areas.

28: The first component in the similarity metric measures how likely it is that $\mathbf{x}_i$ would choose $\mathbf{x}_j$ as its neighbour if neighbours were sampled from a Gaussian centered at $\mathbf{x}_i$, for all i, j.

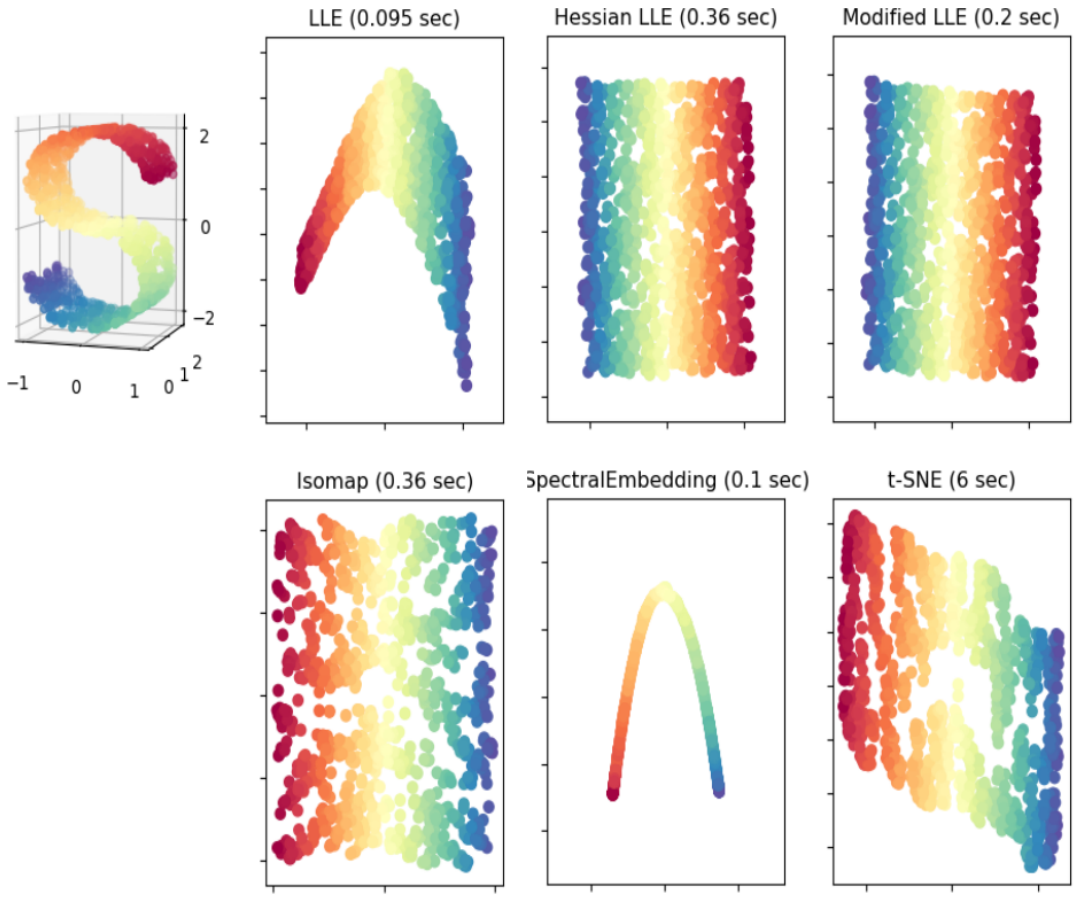


Figure 23.15: Comparison of manifold learning methods on an artificial dataset [20].

The lower-dimensional manifold $\{\mathbf{y}_i\}_{i=1}^n \subseteq \mathcal{M} \subseteq \mathbb{R}^d$ is selected in such a way as to preserve the similarities $p_{i,j}$ as much as possible; this can be achieved by building the (reduced) probabilities

$$q_{i,j} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_{k \neq i}(1 + \|\mathbf{y}_i - \mathbf{y}_k\|^2)^{-1}}$$

for all i, j (note the asymmetry) and minimizing the **Kullback-Leibler divergence** of Q from P:

$$\mathrm{KL}(P||Q) = \sum_{i \neq j} p_{i,j} \log \frac{p_{i,j}}{q_{i,j}}$$

over possible coordinates $\{\mathbf{y}_i\}_{i=1}^n$ [31].

MNIST Example In [20], the methods above are used to learn $2D$ manifolds for the MNIST dataset [13], a database of handwritten digits. The results for 4 of those are shown in Figure 23.17. The analysis of optimal manifold learning methods remains fairly subjective, as it depends not only on the outcome, but also on how much computing power is used and how long it takes to obtain the mapping.

Naïvely, one would expect to see the coordinates in the reduced manifold congregate in 10 (or more) distinct groups; in that regard, $t-$SNE seems to perform admirably on MNIST.

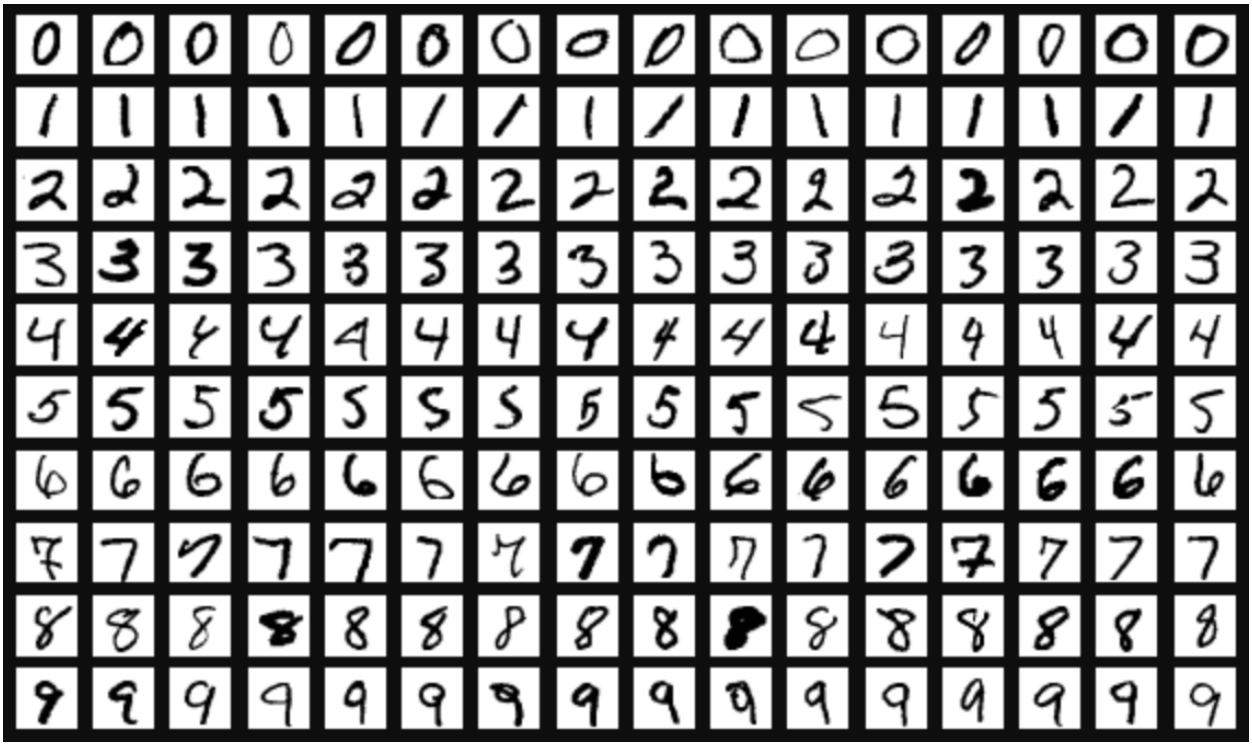


Figure 23.16: Sample of the MNIST dataset [20, 13].

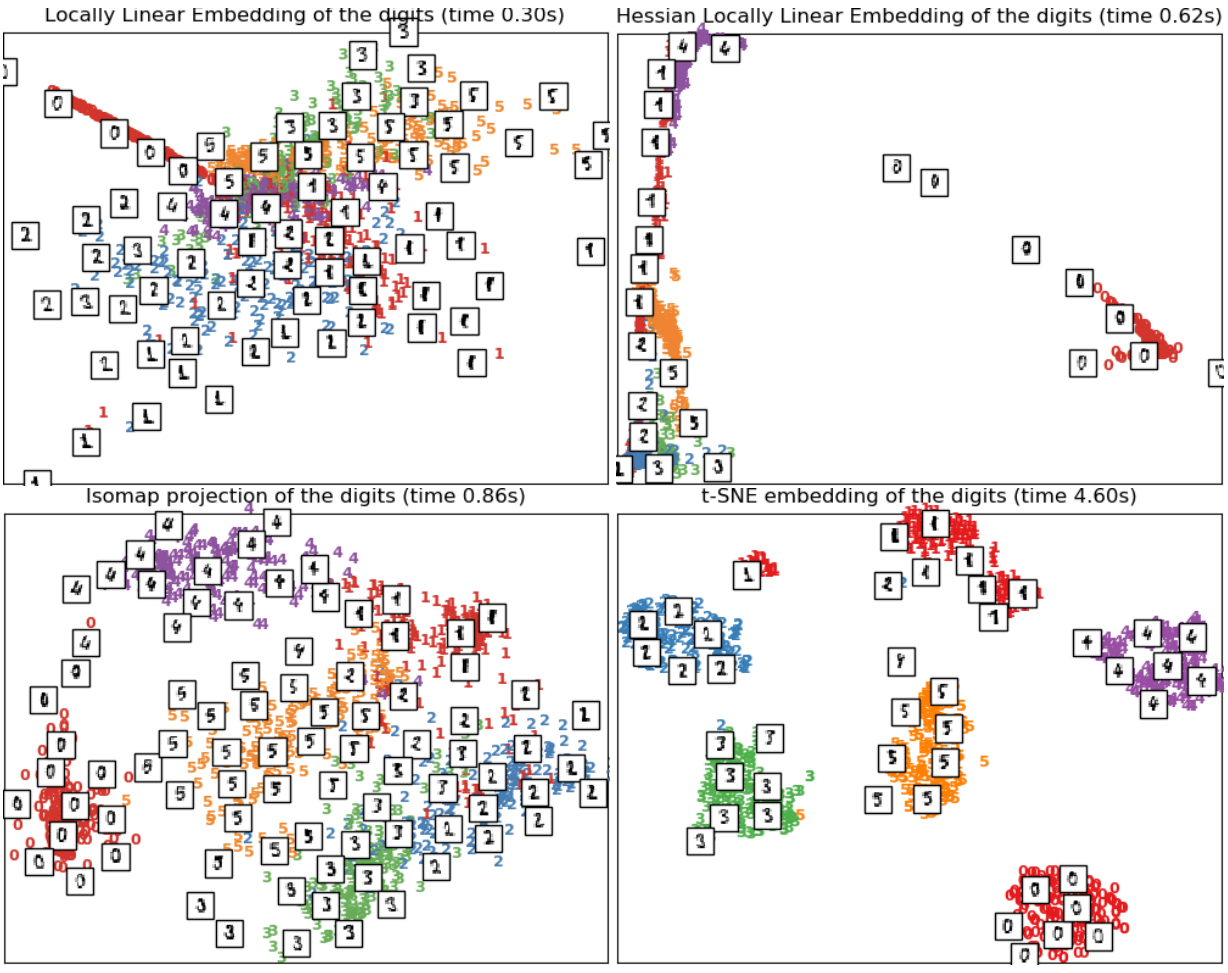


Figure 23.17: Manifold learning on the $0-5$ subset of MNIST: LLE, Hessian LLE, Isomap, $t-$SNE [20, 19].

23.3 Feature Selection

As seen in the previous section, dimension reduction methods can be used to learn low-dimensional manifolds for high-dimensional data, with the hope that the resulting loss in information content can be kept small. Unfortunately, this is not always feasible.

There is a non-technical, yet more problematic, issue with manifold learning techniques: the reduction often fails to provide an easily **interpretable** set of coordinates in the context of the original dataset.

For instance, in a dataset with the 4 features

$$X_1 = \text{Age},\ X_2 = \text{Height},\ X_3 = \text{Weight},\ \text{and}\ X_4 = \text{Rural}\ \in \{0, 1\},$$

say, it is straightforward to **justify** a data-driven decision based on the rule $X_1 = \text{Age} > 25$, for example, but perhaps substantially harder to do so for a reduced rule such as

$$Y_2 = 3(\text{Age} - \overline{\text{Age}}) - (\text{Height} - \overline{\text{Height}}) + 4(\text{Weight} - \overline{\text{Weight}}) + \text{Rural} > 7,$$

even if there is nothing wrong with the rule from a technical perspective.

Furthermore, datasets often contain **irrelevant** and/or **redundant** features; identifying and removing these variables is a common data processing task. The motivations for doing so are varied, but usually fall into one of two categories:

- the modeling tools do not handle redundant variables well, due to **variance inflation** or similar issues, and
- as an attempt by analysts to overcome the **curse of dimensionality** or to avoid situations where the number of variables is larger than the number of observations.

In light of the comment above, the goal of **feature selection** is to remove (and not to transform or project) any attribute that adds noise and reduces the performance of a model, that is to say, to retain a subset of the most **relevant features**[29] , which can help create simpler models, decrease a statistical learner's training time, and reduce overfitting.

29: This usually requires there to be a value to predict, against which the features can be evaluated for relevance; this is discussed further in Chapters 20 and 21.

There are various feature selection methods, typically falling in one of three families – **filter** methods, **wrapper** methods, and **embedded** methods (the next two sections are inspired by [2]):

- **filter methods** focus on the relevance of the features, applying a specific **ranking metric** to each feature, individually. The variables that do not meet a preset benchmark[30] are then removed from consideration, yielding a subset of the most relevant features according to the selected ranking metric; different metrics, and different thresholds, might retain different relevant features;
- **wrapper methods** focus on the usefulness of each feature to the task of interest (usually classification or regression), but do not consider features individually; rather, they evaluate and compare the performance of different combinations of features in order to select the best-performing subset of features, and
- **embedded methods** are a combination of both, using implicit metrics to evaluate the performance of various subsets.

30: Either a threshold on the ranking or on the ranking metric value itself.

Feature selection methods can also be categorized as **unsupervised** or **supervised**:

- **unsupervised methods** determine the importance of features using only their values (with potential feature interactions), while
- **supervised methods** evaluate each feature's importance in relationship with the **target feature**.

Wrapper methods are typically supervised. Unsupervised filter methods search for **noisy** features and include the removal of **constant** variables, of **ID-like** variables (i.e. different on all observations), and of features with **low variability**.

23.3.1 Filter Methods

Filter methods evaluate features without resorting to the use of a classification/regression algorithm. These methods can either be

- **univariate**, where each feature is ranked independently, or
- **multivariate**, where features are considered jointly.

A filter criterion is chosen based on which metric suits the data or problem best.[31] The selected criterion is used to assign a score to, and rank, the features which are then retained or removed in order to yield a **relevant** subset of features.

31: This can be quite difficult to determine.

Features whose score lies above (or below, as the case may be) some pre-selected threshold τ are retained (or removed); alternatively, features whose rank lies below (or above as the case may be) some pre-selected threshold ν are retained (or removed).

Such methods are advantageous in that they are computationally efficient. They also tend to be robust against overfitting as they do not incorporate the effects of the feature subset selection on classification/regression performance.

There are a number of commonly-used filter criteria, including the **Pearson correlation coefficient**, **information gain** (or mutual information), and **relief** [2].

Throughout, let Y be the target variable (assuming that there is one), and $X_1, \ldots, X_p$ be the predictors.

Pearson Correlation Coefficient This value quantifies the linear relationship between two continuous variables [29].

For a predictor X_i, the **Pearson correlation coefficient** between X_i and Y is

$$\rho_i = \frac{\mathrm{Cov}(X_i, Y)}{\sigma_{X_i}\sigma_Y}.$$

Features for which $|\rho_i|$ is **large** (near 1) are linearly (or anti-) correlated with Y, those for which $|\rho_i| \approx 0$ are not linearly (nor anti-linearly) correlated with Y.[32]

32: Which could mean that they are uncorrelated with Y, or that the correlation is not linear or anti-linear.

Only those features with (relatively) strong linear (or anti-linear) correlation are retained.

This correlation ρ_i is only defined if **both** the predictor X_i and the outcome Y are **numerical**; there are alternatives for categorical X_i and Y, or mixed categorical-numerical X_i and Y [30, 9, 12].

In order to get a better handle on what filter feature selection looks like in practice, consider the *Global Cities Index* dataset [26], which ranks prominent cities around the globe on a general scale of "Alpha", "Beta", "Gamma", and "Sufficiency" (1, 2, 3, 4, respectively).

This dataset contains geographical, population, and economics data for 68 ranked cities.

```
globalcities <- data.frame(read.csv("globalcities.csv",
    stringsAsFactors = TRUE))
colnames(globalcities)[2:7] <- c("City.Area","Metro.Area",
    "City.Pop","Metro.Pop", "Ann.Pop.Growth", "GDPpc")
colnames(globalcities)[12:17] <- c("Higher.Ed.Insts",
    "Life.Exp.M","Life.Exp.F","Hospitals", "Museums",
    "Air.Quality")
str(globalcities)
```

```
'data.frame':   68 obs. of  18 variables:
 $ Rating           : int  1 3 2 1 1 1 2 1 2 1 ...
 $ City.Area        : num  165 30.7 38.9 1569 102.6 ...
 $ Metro.Area       : num  807 25437 381 7762 3236 ...
 $ City.Pop         : num  0.76 3.54 0.66 5.72 1.62 ...
 $ Metro.Pop        : num  1.4 4.77 4.01 6.5 3.23 ...
 $ Ann.Pop.Growth   : num  0.01 0.26 0 0.03 0.01 0.04 0 0.01 0.01 0.01 ...
 $ GDPpc            : num  46 21.2 30.5 23.4 36.3 20.3 33.3 15.9 69.3 45.6 ...
 $ Unemployment.Rate: num  0.05 0.12 0.16 0.02 0.15 0.01 0.16 0.1 0.07 0.16 ...
 $ Poverty.Rate     : num  0.18 0.2 0.2 0 0.2 0.01 0.22 0.22 0.17 0.26 ...
 $ Major.Airports   : int  1 1 1 2 1 1 2 1 1 2 ...
 $ Major.Ports      : int  1 0 1 1 1 0 2 0 1 1 ...
 $ Higher.Ed.Insts  : int  23 10 21 37 8 89 30 19 35 25 ...
 $ Life.Exp.M       : num  76.3 75.3 78 69 79 79 82 74.6 74.8 77 ...
 $ Life.Exp.F       : num  80.8 80.8 83.7 74 85.2 83 88 79.7 81.1 82.6 ...
 $ Hospitals        : int  7 7 23 173 45 551 79 22 12 43 ...
 $ Museums          : int  68 36 47 27 69 156 170 76 30 25 ...
 $ Air.Quality      : int  24 46 41 54 35 121 26 77 17 28 ...
 $ Life.Expectancy  : num  78.5 78 80.8 71.5 82.1 ...
```

The `R` package `FSelector` contains feature selection tools, including various filter methods (such as chi-squared score, consistency, various entropy-based filters, etc.). Using its filtering functions, we extract the most relevant features to the ranking of a global city (we treat the `Rating` variable as a numerical response: is this justifiable?).

For instance, if we retain the 5 top predictors for **linear correlation** (Pearson's correlation coefficient) with the response `Rating`, we obtain:

```
(lincor <- FSelector::linear.correlation(
    formula = 'Rating' ~ ., data = globalcities))
```

```
                  attr_importance
City.Area              0.0007479646
Metro.Area             0.0055023564
City.Pop               0.1196632421
Metro.Pop              0.2030923952
Ann.Pop.Growth         0.1935336738
GDPpc                  0.2090866065
Unemployment.Rate      0.2173999333
Poverty.Rate           0.0536585758
Major.Airports         0.2263265771
Major.Ports            0.0563507487
Higher.Ed.Insts        0.0547453393
Life.Exp.M             0.1302972404
Life.Exp.F             0.1412093767
Hospitals              0.1195832079
Museums                0.1553072283
Air.Quality            0.1382099362
Life.Expectancy        0.1380603739
```

```
(subset_lincor <- FSelector::cutoff.k(lincor, k = 5))
```

```
[1] "Major.Airports"    "Unemployment.Rate"    "GDPpc"
[4] "Metro.Pop"         "Ann.Pop.Growth"
```

According to the **linear correlation** feature selection method, the 5 "best" features that relate to a city's global ranking are the number of major airports it has, its unemployment rate, its GDP per capita, its metropolitan population, and its annual population growth.[33]

33: As filtering is a **pre-processing step**, proper analysis would also require building a model using this subset of features.

Mutual Information **Information gain** is a popular **entropy-based** method of feature selection that measures the amount of dependence between features by quantifying the amount of mutual information between them. In general, this quantifies the **amount of information** obtained about a predictor X_i by observing the target feature Y.

Mutual information can be expressed as

$$\text{IG}(X_i; Y) = H(X_i) - H(X_i \mid Y),$$

where $H(X_i)$ is the **marginal entropy** of X_i and $H(X_i \mid Y)$ is the **conditional entropy** of X_i given Y [28], an

$$H(X_i) = \text{E}_{X_i}[-\log p(X_i)], \quad H(X_i \mid Y) = \text{E}_{(X_i,Y)}[-\log p(X_i \mid Y)],$$

where $p(X_i)$ and $p(X_i \mid Y)$ are the probability density functions of the random variables X_i and $X_i \mid Y$, respectively.

How is IG interpreted? Consider the following example: let Y represent the salary of an individual (continuous), X_1 their hair colour (categorical), X_2 their age (continuous), X_3 their height (continuous), and X_4 their self-reported gender (categorical). A sample of $n = 2144$ individuals is found in `demo1.csv`.

```
salary <- data.frame(read.csv("demo1.csv",
    stringsAsFactors = TRUE))
colnames(salary)[1] <- c("Hair")
```

Some summary statistics are shown below:

Hair	
Black	1813
Blonde	58
Brown	221
Red	52
	2144

Gender	
Female	1083
Male	1061
	2144

Summary	Age	Height	Salary
MIN	16	136	8000
Q1	29	164	34000
MED	40	172	48000
Q3	53	179	61250
MAX	65	208	103000
MEAN	40.4	171.6	47674.4
STDEV	14.2	11.2	19710.3
SKEW	0.01	0.06	-0.04

Salary	Hair				Total
Deciles	Black	Blonde	Brown	Red	
1	193	7	25	8	233
2	180	9	25	8	222
3	180	4	30	6	220
4	187	4	14	3	208
5	182	8	26	8	224
6	185	5	18	1	209
7	183	5	18	5	211
8	184	6	19	8	217
9	168	8	22	5	203
10	171	2	24		197
Total	**1813**	**58**	**221**	**52**	**2144**

Salary	Age Deciles										Total
Deciles	1	2	3	4	5	6	7	8	9	10	
1	203	30									233
2	40	133	39	1	1	5	1			2	222
3		26	73	21	24	15	11	4	14	32	220
4		6	26	50	27	17	13	14	24	31	208
5		5	31	44	31	18	16	25	26	28	224
6		3	21	39	34	27	16	21	33	15	209
7			7	40	33	36	23	31	19	22	211
8			2	30	38	36	28	33	20	30	217
9				2	19	54	30	39	36	23	203
10				2	12	41	32	57	47	6	197
Total	**243**	**203**	**199**	**229**	**219**	**249**	**170**	**224**	**219**	**189**	**2144**

Salary	Height Deciles										Total
Deciles	1	2	3	4	5	6	7	8	9	10	
1	28	22	21	24	25	17	27	21	21	27	233
2	19	26	22	18	30	19	23	22	22	21	222
3	34	38	34	26	39	12	15	9	6	7	220
4	33	32	31	28	18	16	15	20	8	7	208
5	34	29	19	25	32	18	21	20	14	12	224
6	33	25	26	14	33	13	22	15	13	15	209
7	20	24	24	17	24	16	16	19	30	21	211
8	12	15	15	22	32	16	18	36	20	31	217
9	6	13	12	17	27	9	24	28	34	33	203
10	1	4	10	15	18	13	30	35	31	40	197
Total	**220**	**228**	**214**	**206**	**278**	**149**	**211**	**225**	**199**	**214**	**2144**

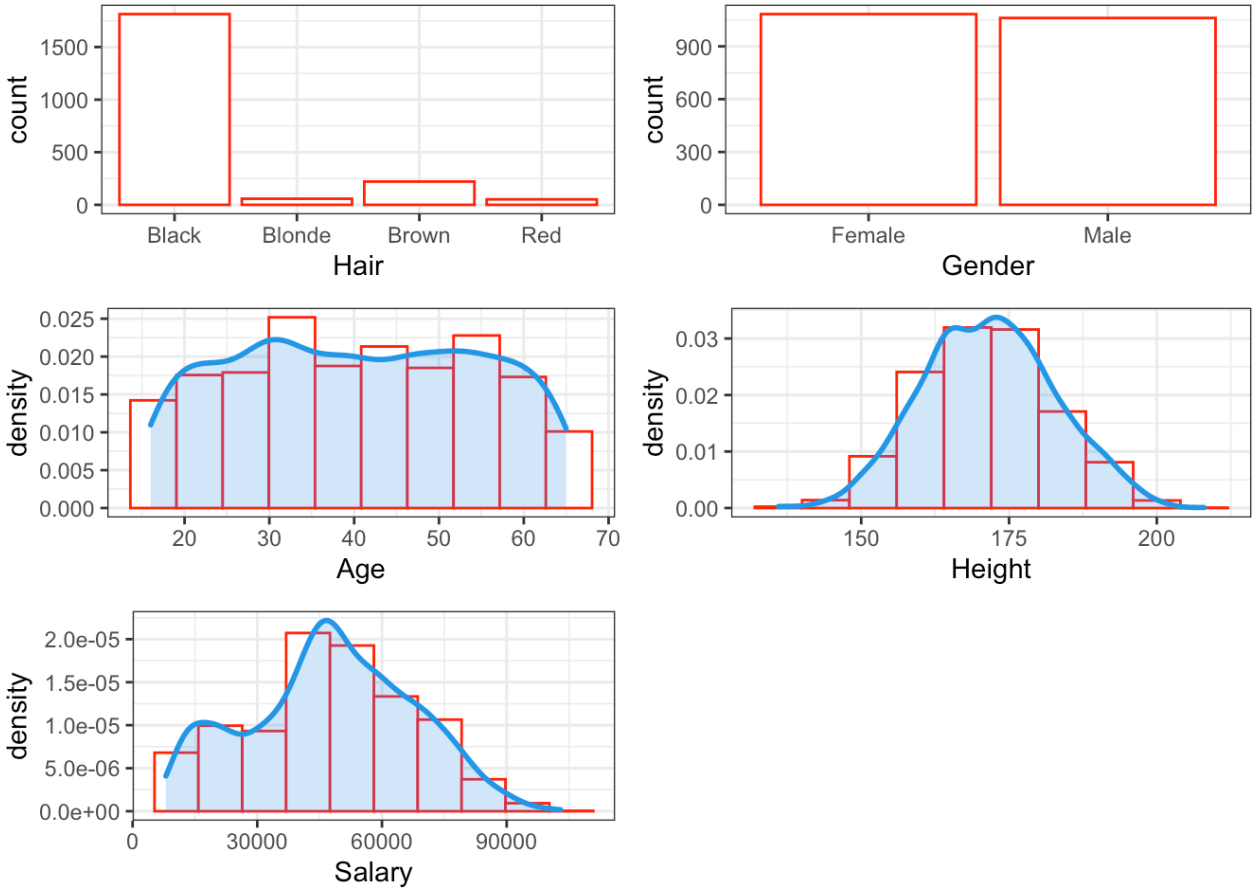
Salary	Gender		Total
Deciles	Female	Male	
1	144	89	233
2	117	105	222
3	198	22	220
4	169	39	208
5	158	66	224
6	128	81	209
7	95	116	211
8	54	163	217
9	18	185	203
10	2	195	197
Total	**1083**	**1061**	**2144**

In a general population, one would expect that the distribution of salaries, among others, is likely to be fairly haphazard, and it might be hard to explain why it has the shape that it does, specifically.

Distributions of the predictors and the response

```
library(ggplot2)
par(mfrow=c(3,2))
plot1 <- ggplot(salary, aes(x=Hair)) +
  geom_bar(color='red', fill='white') +
  theme_bw()

plot2 <- ggplot(salary, aes(x=Gender)) +
  geom_bar(color='red', fill='white') +
  theme_bw()

plot3 <- ggplot(salary, aes(x=Age)) +
  geom_histogram(aes(y=..density..),
  color='red', fill='white', bins=10) +
  geom_density(lwd = 1, colour = 4, fill = 4, alpha = 0.25) +
  theme_bw()

plot4 <- ggplot(salary, aes(x=Height)) +
  geom_histogram(aes(y=..density..),
```

```
  color='red', fill='white', bins=10) +
  geom_density(lwd = 1, colour = 4, fill = 4, alpha = 0.25) +
  theme_bw()

plot5 <- ggplot(salary, aes(x=Salary)) +
  geom_histogram(aes(y=..density..),
  color='red', fill='white', bins=10) +
  geom_density(lwd = 1, colour = 4, fill = 4, alpha = 0.25) +
  theme_bw()

gridExtra::grid.arrange(plot1, plot2, plot3, plot4, plot5, ncol=2)
```



Perhaps it could be explained by knowing the relationship between the salary and the other variables? It is this idea that forms the basis of mutual information feature selection.

Applying the definition, one sees that

$$H(X_1) = -\sum_{\text{colour}} p(\text{colour}) \log p(\text{colour})$$

$$H(X_2) = -\int p(\text{age}) \log p(\text{age}) \, d\text{age}$$

$$H(X_3) = -\int p(\text{height}) \log p(\text{height}) \, d\text{height}$$

$$H(X_4) = -\sum_{\text{gender}} p(\text{gender}) \log p(\text{gender})$$

$$H(X_1 \mid Y) = -\int p(Y) \left\{ \sum_{\text{colour}} p(\text{colour} \mid Y) \log p(\text{colour} \mid Y) \right\} dY$$

$$H(X_2 \mid Y) = -\iint p(Y) p(\text{age} \mid Y) \log p(\text{age} \mid Y)\, d\text{age}\, dY$$

$$H(X_3 \mid Y) = -\iint p(Y) p(\text{ht} \mid Y) \log p(\text{ht} \mid Y)\, d\text{ht}\, dY$$

$$H(X_4 \mid Y) = -\int p(Y) \left\{ \sum_{\text{gender}} p(\text{gender} \mid Y) \log p(\text{gender} \mid Y) \right\} dY$$

If the theoretical distributions are known, the entropy integrals can be computed directly (or approximated using standard numerical integration methods).

Gender and hair colour can be fairly easily be modeled using multinomial distributions, but there is more uncertainty related to the numerical variables. A potential approach is to recode the continuous variables age, height, and salary as **decile variables** a_d, h_d, and Y_d taking values $\{1, \ldots, 10\}$ according to which decile of the original variable the observation falls (see decile breakdown above).

The integrals can then be replaced by sums:

$$H(X_1) = -\sum_{\text{colour}} p(\text{colour}) \log p(\text{colour})$$

$$H(X_2) \approx -\sum_{k=1}^{10} p(\text{a}_d = k) \log p(\text{a}_d = k)$$

$$H(X_3) \approx -\sum_{k=1}^{10} p(\text{ht}_d = k) \log p(\text{ht}_d = k)$$

$$H(X_4) = -\sum_{\text{gender}} p(\text{gender}) \log p(\text{gender})$$

$$H(X_1 \mid Y) \approx -\sum_{j=1}^{10} p(Y_d = j) \sum_{c \in \text{colour}} p(c \mid Y_d = j) \log p(c \mid Y_d = j)$$

$$H(X_2 \mid Y) \approx -\sum_{j=1}^{10} p(Y_d = j) \sum_{k=1}^{10} p(a_d = k \mid Y_d = j) \log p(a_d = k \mid Y_d = j)$$

$$H(X_3 \mid Y) \approx -\sum_{j=1}^{10} p(Y_d = j) \sum_{k=1}^{10} p(h_d = k \mid Y_d = j) \log p(h_d = k \mid Y_d = j)$$

$$H(X_4 \mid Y) \approx -\sum_{j=1}^{10} p(Y_d = k) \sum_{g \in \text{gender}} p(g \mid Y_d = j) \log p(g \mid Y_d = j)$$

The results are shown below (using base 10 logarithms, and rounded out to the nearest hundredth):

X	H(X)	H(X | Y)	IG(X;Y)	Ratio
Hair	0.24	0.24	0.00	0.00
Age	1.00	0.74	0.26	0.26
Height	1.00	0.96	0.03	0.03
Gender	0.30	0.22	0.08	0.26

The percentage decrease in entropy after having observed Y is shown in the column "Ratio." Raw IG numbers would seem to suggest that Gender has a small link to Salary; the Ratio numbers suggest that this could be due to the way the Age and Height levels have been categorized (as deciles).

Relief This approach scores (numerical) features based on the identification of feature value differences between nearest-neighbour instance pairs.

If there is a feature value difference in a neighbouring instance pair **of the same class**, the score of the feature decreases; on the other hand, if there exists a feature value difference in a neighbouring instance pair with **different class values**, the feature score increases.

More specifically, let

$$D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n \subset \mathbb{R}^p \times \{\pm 1\}$$

be a dataset where $\mathbf{x}_i$ is the i-th data sample and y_n is its corresponding class label.

For each feature j and observation i, two values are selected: the **near hit** $H(x_{i,j})$ is the value of X_j which is nearest to $x_{i,j}$ among all instances in the same class as $\mathbf{x}_i$, while the **near miss** $M(x_{i,j})$ is the value of X_j which is nearest to $x_{i,j}$ among all instances in the opposite class of $\mathbf{x}_i$.

The **Relief score** of the j^{th} feature is

$$S_j^d = \frac{1}{n} \sum_{i=1}^n \left\{ d(x_{i,j}, M(x_{i,j})) - d(x_{i,j}, H(x_{i,j})) \right\},$$

for some pre-selected distance $d : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_0^+$.

A feature for which near-hits tend to be nearer to their instances than near-misses are (i.e., for which

$$d(x_{i,j}, M(x_{i,j})) > d(x_{i,j}, H(x_{i,j})),$$

on average) will yield **larger** Relief scores than those for which the opposite is true. Features are deemed **relevant** when their relief score is greater than some threshold τ.

There are a variety of Relief-type measurements to accommodate potential feature interactions or multi-class problems[34] (ReliefF), but in general Relief is noise-tolerant and robust to interactions of attributes; its effectiveness decreases for small training sets, however [23].

The Relief algorithm is also implemented in the `R` package `CORElearn`, as are numerous other methods:

```
CORElearn::infoCore(what="attrEval") # Classification
```

```
[1] "ReliefFequalK"      "ReliefFexpRank"     "ReliefFbestK"
[4] "Relief"             "InfGain"            "GainRatio"
[7] "MDL"                "Gini"               "MyopicReliefF"
```

34: For instance, for a $p-$distance δ, set

$$H^\delta(x_{i,j}) = \arg\min_{\pi_j(\mathbf{z})} \{\delta(\mathbf{x}_i, \mathbf{z}) \mid \mathscr{C}(\mathbf{x}_i) = \mathscr{C}(\mathbf{z})\}$$

and

$$M^\delta(x_{i,j}) = \arg\min_{\pi_j(\mathbf{z})} \{\delta(\mathbf{x}_i, \mathbf{z}) \mid \mathscr{C}(\mathbf{x}_i) \neq \mathscr{C}(\mathbf{z})\}.$$

```
[10] "Accuracy"           "ReliefFmerit"       "ReliefFdistance"
[13] "ReliefFsqrDistance" "DKM"                "ReliefFexpC"
[16] "ReliefFavgC"        "ReliefFpe"          "ReliefFpa"
[19] "ReliefFsmp"         "GainRatioCost"      "DKMcost"
[22] "ReliefKukar"        "MDLsmp"             "ImpurityEuclid"
[25] "ImpurityHellinger"  "UniformDKM"         "UniformGini"
[28] "UniformInf"         "UniformAccuracy"    "EqualDKM"
[31] "EqualGini"          "EqualInf"           "EqualHellinger"
[34] "DistHellinger"      "DistAUC"            "DistAngle"
[37] "DistEuclid"
```

```
CORElearn::infoCore(what="attrEvalReg") # Regression
```

```
[1] "RReliefFequalK"      "RReliefFexpRank"     "RReliefFbestK"
[4] "RReliefFwithMSE"     "MSEofMean"           "MSEofModel"
[7] "MAEofModel"          "RReliefFdistance"    "RReliefFsqrDistance"
```

Again working on the *Global Cities Dataset*, we start by declaring the target variable `Rating` as a categorical variable.

```
globalcities.cat <- globalcities
globalcities.cat$Rating <- as.factor(globalcities.cat$Rating)
```

Now, let's evaluate the predictors relevance, using `InfGain` and `ReliefFpe`, say:

```
InfGain.wts <- CORElearn::attrEval(
    Rating ~ ., globalcities.cat, estimator="InfGain")
ReliefF.wts <- CORElearn::attrEval(
    Rating ~ ., globalcities.cat, estimator="ReliefFpe")
data.frame(InfGain.wts,ReliefF.wts)
```

```
                 InfGain.wts ReliefF.wts
City.Area          0.05898196 0.035227591
Metro.Area         0.10501737 0.020691027
City.Pop           0.10422894 0.050779784
Metro.Pop          0.13022307 0.006038070
Ann.Pop.Growth     0.04624721 0.005785050
GDPpc              0.12909419 0.038870952
Unemployment.Rate  0.08824268 0.069113116
Poverty.Rate       0.06966089 0.004854548
Major.Airports     0.03565266 0.015932195
Major.Ports        0.04175416 0.014967595
Higher.Ed.Insts    0.02868171 0.008031587
Life.Exp.M         0.09504723 0.119356004
Life.Exp.F         0.04937724 0.073946982
Hospitals          0.18080212 0.011268651
Museums            0.10732739 0.017808737
Air.Quality        0.05838298 0.057001715
Life.Expectancy    0.07730300 0.101021497
```

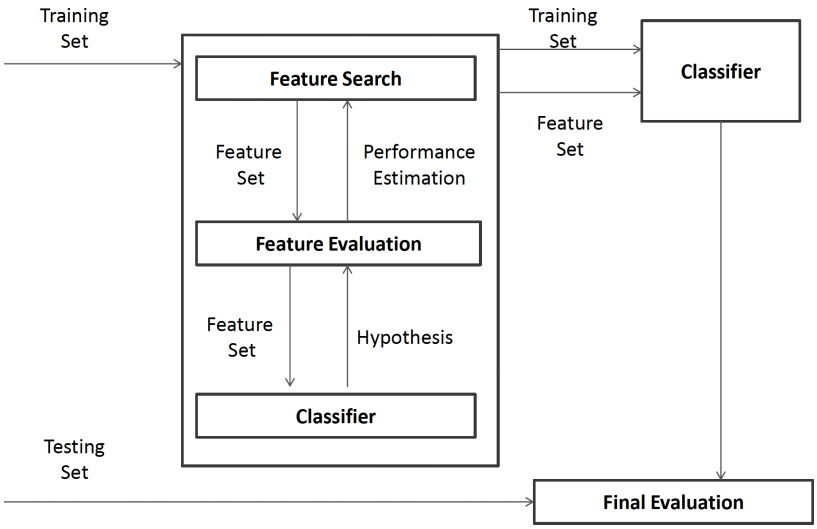


Figure 23.18: Feature selection process for wrapper methods in classification problems [2].

For classification tasks (categorical targets), the more relevant features are those for which the scores are **higher**.[35] How do the top-5 for each method compare in the previous example? Should this be surprising?

35: This is not necessarily the case for regression tasks, however – be sure to read the documentation for each method.

There is a multitude of other filter methods, including [2, 3]:

- **correlation metrics** (Kendall, Spearman, point-biserial, etc.);
- **entropy-based metrics** (gain ratio, symmetric uncertainty, etc.);
- **relief-type algorithms** (ReliefF, Relieved-F, etc.);
- χ^2-test;
- ANOVA;
- Fisher Score;
- Gini Index;
- etc.

The list is by no means exhaustive, but it provides a fair idea of the various types of filter-based feature selection metrics.

23.3.2 Wrapper Methods

Wrapper methods offer a powerful way to address problem of variable selection. Wrapper methods evaluate the quality of subsets of features for predicting the target output under the selected predictive algorithm and select the optimal combination (for the given training set and algorithm).

In contrast to filter methods, wrapping methods are integrated directly into the classification or clustering process (see Figure 23.18 for an illustration of this process).

Wrapper methods treats feature selection as a **search problem** in which different subsets of features are explored. This process can be computationally expensive as the size of the search space increases exponentially with the number of predictors; even for modest problems an exhaustive search can quickly become impractical.

In general, wrapper methods iterate over the following steps, until an "optimal" set of features is identified:

- select a feature subset, and
- evaluate the performance of the selected feature subset.

The search ends once some desired quality is reached (such as adjusted R^2, accuracy, etc.). Various search methods are used to traverse the feature phase space and provide an approximate solution to the **optimal feature subset problem**, including: hill-climbing, best-first, and genetic algorithms.

Wrapper methods are not as efficient as filter methods and are not as robust against over-fitting. However, they are very effective at improving the model's performance due to their attempt to minimize the error rate.[36]

36: Which unfortunately can also lead to the introduction of implicit bias in the problem [2].

23.3.3 Subset Selection Methods

Stepwise selection is a form of *Occam's Razor*: at each step, a new feature is considered for inclusion or removal from the current features set based on some criterion ($F-$test, $t-$test, etc.). Greedy search methods such as backward elimination and forward selection have proven to be both quite robust against over-fitting and among the least computationally expensive wrapper methods.

Backward elimination begins with the full set of features and sequentially eliminates the least relevant ones until further removals increase the error rate of the predictive model above some utility threshold.

Forward selection works in reverse, beginning the search with an empty set of features and progressively adding relevant features to the ever growing set of predictors. In an ideal setting, model performance should be tested using **cross-validation**.

Stepwise selection methods are extremely common, but they have severe limitations (which are not usually addressed) [7, 10]:

- the tests are biased, since they are all based on the same data;
- the adjusted R^2 only takes into account the number of features in the final fit, and not the degrees of freedom that have been used in the entire model;
- if cross-validation is used, stepwise selection has to be repeated for each sub-model but that is not usually done, and
- it represents a classic example of p-hacking.

Consequently, the use of stepwise methods is **contra-indicated** in the machine learning context.

23.3.4 Regularization (Embedded) Methods

An interesting hybrid is provided by the **least absolute shrinkage and selection operator** (LASSO) and its variants, which are discussed in Section 20.2.4.

23.3.5 Supervised and Unsupervised Feature Selection

While feature selection methods are usually categorised as filter, wrapper, or embedded, they can also be categorised as **supervised** or **unsupervised** methods. Whether a feature selection method is supervised or not boils down to whether the labels of objects/instances are incorporated into the feature reduction process (or not).

The methods that have been described in this section were all supervised.

In unsupervised methods, feature selection is carried out based on the characteristics of the attributes, without any reference to labels or a target variable. In particular, for **clustering problems** (see Section 22), only unsupervised feature selection methods can be used [1].

Unsupervised feature selection methods include:

- identifying ID-like predictors;
- identifying constant (or nearly constant) predictors;
- identifying predictors that are in a multicolinear relationship with other variables;
- identifying clusters of predictors, etc.

23.4 Advanced Topics

When used appropriately, the approaches to feature selection and dimension reduction methods presented in the last two sections provide a solid toolkit to help mitigate the effects of the curse of dimensionality.

However, they remain (for the most part) rather straightforward. The methods discussed in this section are decidedly more sophisticated, from a mathematical perspective; an increase in conceptual complexity can lead to insights that are out of reach of more direct approaches.

23.4.1 Singular Value Decomposition

From a database management perspective, it pays not to view datasets simply as flat file arrays; from an analytical perspective, however, viewing datasets as **matrices** allows analysts to use the full machinery of linear algebra and **matrix factorization** techniques, of which **singular value decomposition** (SVD) is a well-known component.[37]

37: Matrix factorization techniques have applications to other data analytic tasks; notably, they can be used to impute missing values and to build recommender systems.

As before, let $\{\mathbf{x}_i\}_{i=1}^n \subseteq \mathbb{R}^p$ and denote the **matrix of observations** by

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \in \mathbb{M}_{n,p}(\mathbb{R}) = \mathbb{R}^{n \times p}.$$

Let $d \geq \min p, n$. From a dimension reduction perspective, the goal of matrix factorization is to find two narrow matrices $\mathbf{W}_d \in \mathbb{R}^{n \times d}$ (the **cases**)

and $\mathbf{C}_d \in \mathbb{R}^{p\times d}$ (the **concepts**) such that the product $\mathbf{W}_d\mathbf{C}_d^\top = \widetilde{\mathbf{X}}_d$ is the best rank d approximation of $\mathbf{X}$, i.e.

$$\widetilde{\mathbf{X}}_d = \arg\min_{\mathbf{X}'}\{\|\mathbf{X}-\mathbf{X}'\|_F\}, \quad \text{with } \operatorname{rank}(\mathbf{X}') = d,$$

where the **Frobenius norm** F of a matrix is

$$\|\mathbf{A}\|_F^2 = \sum_{i,j}|a_{i,j}|^2.$$

In a sense, $\widetilde{\mathbf{X}}_d$ is a **"smooth" representation** of $\mathbf{X}$; the dimension reduction takes place when $\mathbf{W}_d$ is used as a dense $d-$representation of $\mathbf{X}$.The link with the singular value decomposition of $\mathbf{X}$ can be made explicit: there exist orthonormal matrices $\mathbf{U} \in \mathbb{R}^{n\times n}$, $\mathbf{V} \in \mathbb{R}^{p\times p}$, and a diagonal matrix $\mathbf{\Sigma} \in \mathbb{R}^{n\times p}$ with $\sigma_{i,j} = 0$ if $i \neq j$ and $\sigma_{i,i} \geq \sigma_{i+1,i+1} \geq 0$ for all i,[38] such that

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top;$$

the decomposition is unique if and only if $n = p$.

38: Each **singular value** is the principal square root of the corresponding eigenvalue of the covariance matrix $\mathbf{X}^\top\mathbf{X}$ (see Section 23.2.3).

Let $\mathbf{\Sigma}_d \in \mathbb{R}^{d\times d}$ be the matrix obtained by retaining the first d rows and the the first d columns of $\mathbf{\Sigma}$; $\mathbf{U}_d \in \mathbb{R}^{n\times d}$ be the matrix obtained by retaining the first d columns of $\mathbf{U}$, and $\mathbf{V}_d^\top \in \mathbb{R}^{d\times p}$ be the matrix obtained by retaining the first d rows of $\mathbf{V}^\top$.

Then

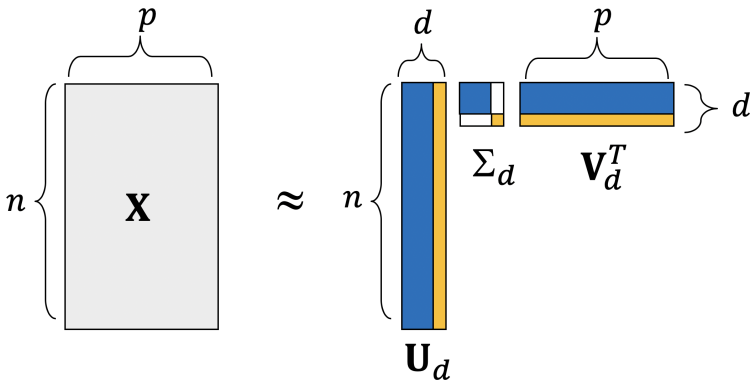
$$\widetilde{\mathbf{X}}_d = \underbrace{\mathbf{U}_d\mathbf{\Sigma}_d}_{\mathbf{W}_d}\mathbf{V}_d^\top,$$

and the d-dimensional rows of $\mathbf{W}_d$ are approximations of the $p-$dimensional rows of $\mathbf{X}$ in the sense that

$$\langle \mathbf{W}_d[i], \mathbf{W}_d[j]\rangle = \left\langle \widetilde{\mathbf{X}}_d[i], \widetilde{\mathbf{X}}_d[j]\right\rangle \approx \langle \mathbf{X}_d[i], \mathbf{X}_d[j]\rangle \text{ for all } i, j.$$

Applications

1. One of the advantages of SVD is that it allows for substantial savings in data storage (modified from [14]):



Storing $\mathbf{X}$ requires np saved entries, but an approximate version of the original requires only $d(n+p+d)$ saved entries; if $\mathbf{X}$ represents a 2000×2000 image (with 4 million entries) to be transmitted, say, a decent approximation can be sent *via* $d = 10$ using only 40100

Figure 23.19: SVD image reconstruction: $d = 1400$ (left), $d = 10$ (middle), $d = 50$ (right); Llewellyn and Gwynneth Rayfield.

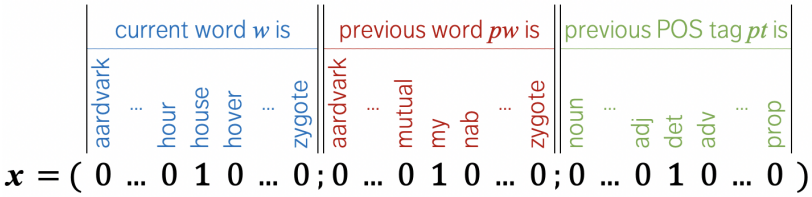
entries, roughly 1% of the original number of entries (see Figure 23.19 for an illustration).

2. SVD can also be used to learn **word embedding vectors**. In the traditional approach to text mining and natural language processing (NLP) (see Sections 27 and 32), words and associated concepts are represented using **one-hot encoding**.[39]

39: Sparse vectors whose entries are 0 or 1, based on the identity of the words and POS tags under consideration.

For instance, if the task is to predict the part-of-speech (POS) tag of a word given its context in a sentence (current and previous word identities w and pw, as well as the latter's part-of-speech (POS) tag pt), the input vector could be obtained by concatenation of the one-hot encoding of w, the one-hot encoding of pw, and the one-hot encoding of pt.

The input vector that would be fed into a classifier to predict the POS of the word "house" in the sentence fragment "my house", say, given that "my" has been tagged as a 'determiner' could be:



The sparsity of this vector is a major CoD liability: a reasonably restrictive vocabulary subset of English might contain $|V_W| \approx 20,000$ words, while the Penn Treebank project recognizes ≈ 40 POS tags, which means that $\mathbf{x} \in \mathbb{R}^{40,040}$ (at least).

Another issue is that the one-hot encoding of words does not allow for meaningful similarity comparisons between words: in NLP, words are considered to be similar if they appear in similar sentence **contexts**.[40]

40: "Ye shall know a word by the company it keeps", as the **distributional semantics** saying goes. The term "kumipwam" is not found in any English dictionary, but its probable meaning as "a small beach/sand lizard" could be inferred from its presence in sentences such as "Elowyn saw a tiny scaly kumipwam digging a hole on the beach". It is easy to come up with examples where the context is ambiguous, but on the whole the contextual approach has proven itself to be mostly reliable.

The terms "black" and "white" are similar in this framework as they both often occur immediately preceding the same noun (such as "car", "dress", etc.); human beings recognize that the similarity goes further than both of the terms being adjectives – they are both colours, and are often used as opposite poles on a variety of spectra. This similarity is impossible to detect from the one-hot encoding vectors, however, as all its word representations are exactly as far from one another.

SVD proposes a single resolution to both of these issues. Let $\mathbf{M}^f \in \mathbb{R}^{|V_W| \times |V_C|}$ be the **word-context** matrix of the association measure f, derived from some appropriate corpus, that is, if $V_W = \{w_1, \ldots, w_{|V_W|}\}$ and $V_C = \{c_1, \ldots, c_{|V_C|}\}$ are the vocabulary and contexts of the corpus, respectively, then $M^f_{i,j} = f(w_i, c_j)$ for all i, j.

For instance, one could have

$$V_W = \{\text{aardvark}, \ldots, \text{zygote}\}\,,$$
$$V_C = \{\ldots, \text{word appearing before "cat"}, \ldots\},$$

and f given by **positive pointwise mutual information** for words and contexts in the corpus (the specifics of which are not germane to the discussion at hand; see [5] for details).

The SVD

$$\mathbf{M}^f \approx \mathbf{M}^f_d = \mathbf{U}_d \mathbf{\Sigma}_d \mathbf{V}^\top_d$$

yields $d-$dimensional word embedding vectors $\mathbf{U}_d\mathbf{\Sigma}_d$ which preserve the **context-similarity** property discussed above. The decomposition of the POS-context matrix, where words are replaced by POS tags, produces POS embedding vectors.

Perhaps a pre-calculated 4-dimensional word embedding of V_W is:

Word Embeddings

aardvark	-0.37	-0.23	0.33	-0.02
⋮			⋮	
hour	0.38	-0.37	-0.21	-0.11
house	**0.31**	**0.12**	**-0.03**	**0.31**
hover	-0.10	0.07	0.37	0.15
⋮			⋮	
mutual	0.26	0.25	-0.39	-0.07
my	**-0.41**	**0.37**	**-0.12**	**0.02**
nab	-0.43	-0.37	-0.12	0.13
⋮			⋮	
zygote	0.06	-0.21	-0.38	-0.28

while a 3-dimensional POS embeddings could be:

POS Embeddings

noun	0.11	-0.21	0.01
⋮		⋮	
adj	0.21	0.88	0.71
det	**0.17**	**0.51**	**0.64**
adv	-0.35	0.37	0.37
⋮		⋮	
prop	-0.01	0.08	0.74

leading to a 11-dimensional representation $\mathbf{x}'$ of $\mathbf{x}$

	current word w is				previous word pw is				previous POS tag pt is		
$\boldsymbol{x}' =$	0.31	0.12	-0.03	**0.81**	-0.41	0.32	-0.12	**0.02**	0.17	0.51	0.64

which provides a **substantial reduction** in the dimension of the input space.

23.4.2 PCA Regression and Partial Least Squares

For $m = 1, \ldots, M \leq p$, we let $z_m = \vec{X}^\top \boldsymbol{\phi}_m$ be linear combinations of the original predictors $\{X_1, \ldots, X_p\}$.

If we are fitting $y = f(\mathbf{x}) = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}]$ using OLS, we can also fit $y_i = \theta_0 + \mathbf{z}_i^\top \boldsymbol{\theta} + \varepsilon_i$, $i = 1, \ldots, n$ using OLS. If the constants $\phi_{m,j}$ are selected **wisely**, then transforming the variables can yield a model that outperforms OLS regression – the predictions might be better than those obtained by fitting $y_i = \beta_0 + \mathbf{x}_i^\top \boldsymbol{\beta} + \varepsilon_i$, $i = 1, \ldots, N$.

By definition, $\theta_0 = \beta_0$ and

$$\begin{aligned}\mathbf{z}_i^\top \boldsymbol{\theta} &= \sum_{m=1}^{M} \theta_m z_{i,m} = \sum_{m=1}^{M} \theta_m \mathbf{x}_i^\top \boldsymbol{\phi} = \sum_{m=1}^{M} \theta_m \sum_{j=1}^{p} \phi_{m,j} x_{i,j} \\ &= \sum_{j=1}^{p} \sum_{m=1}^{M} \theta_m \phi_{m,j} x_{i,j} = \sum_{j=1}^{p} \beta_j x_{i,j} = \mathbf{x}_i^\top \boldsymbol{\beta},\end{aligned}$$

where $\beta_j = \sum_{m=1}^{M} \theta_m \phi_{m,j}$, which is to say that the dimension reduction regression is a special case of the original linear regression model, with constrained coefficients β_j.

Such constraints can help with the bias-variance trade-off (when $p \gg n$, picking $M \ll p$ can reduce the variance of the fitted coefficients).

The challenge then is to find an appropriate way to pick the $\phi_{m,j}$. We will consider two approaches: **principal components** and **partial least squares**.

Principal Components Regression Let us assume that M **principal components** $\{Z_1, \ldots, Z_M\}$ have been retained (see Section 23.2.3), where

$$Z_i = \mathbf{w}_i^\top (X_1, \ldots, X_p),$$

assuming that the eigenvectors $\mathbf{w}_i$ are ordered according to the corresponding eigenvalues ($\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$):

- the first principal component is the **normalized** (centered and scaled) linear combination of variables with the largest variance;
- the second principal component is the normalized linear combination of variables with the largest variance, subject to having no correlation with all previous components (the first);
- ...
- the Mth principal component is the normalized linear combination of variables with the largest variance, subject to having no correlation with all previous components.

The regression function $f(\mathbf{x}) = \mathrm{E}[Y \mid \vec{X} = \mathbf{x}]$ is hopefully well approximated by the function $g(\mathbf{z}) = \mathrm{E}[Y \mid \vec{Z} = \mathbf{z}]$, i.e.,

$$\hat{y}_z = g(\mathbf{z}) = \gamma_0 + \gamma_1 z_1 + \cdots + \gamma_M z_M$$

should compare "reasonably well" to

$$\hat{y}_z = f(\mathbf{z}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p.$$

The main challenge is to determine the optimal M. If M is too small, we run the risk of having a model with high squared bias and low variance (**underfitting**); if M is too large, not only we we not achieve much in the way of dimension reduction, but we might produce a model with low squared bias and high variance (**overfitting**).

Any method that allows for the estimation of MSE_{Te} (such as cross-validation) could be used to select M, but there are other approaches as well (again, see Section 23.2.3).

Partial Least Squares In **principal component regression** (PCR), the identified **directions** (linear combinations) that best represent the predictors $\{X_1, \ldots, X_p\}$ are determined in an **unsupervised** manner: the response Y plays no role in determining the principal components.

As such, there is no guarantee that the directions that best explain the predictors are also the best directions to use to predict the response. The framework for **partial least squares** is the same as that for PCR, except that the directions Z_i are selected both to explain the predictors and to be related to the response Y.

As in PCR, we start by **normalizing** (centering and scaling) the predictor part of the training set Tr. The **first direction** Z_1 is computed using the OLS coefficient estimates of

$$Y_i = \phi^1_{0,j} + \phi_{1,j} X_{i,j} + \gamma_i, \quad j = 1, \ldots, p, i = 1, \ldots, N.$$

Note that each $\phi_{1,j}$ is proportional to $\rho_{X_j,Y}$ and that the direction

$$Z_1 = \sum_{j=1}^{p} \phi_{1,j} X_j = \boldsymbol{\phi}_1^\top \vec{X}$$

places the highest weights on the predictors that are most strongly linked to the response. Now, we run an OLS regression of Y using Z_1 as a predictor:

$$Y_i = \psi_0 + \psi_1 z_{1,i} + \varepsilon_i, \quad i = 1, \ldots, N$$

and let $\varepsilon_i = Y_i - \psi_0 - \psi_1 z_{1,i}$ be the component of the data not "explained" by Z_1.

The **second direction** Z_2 is computed using the OLS coefficient estimates of

$$\varepsilon_i = \phi^2_{0,j} + \phi_{2,j} X_{i,j} + \gamma_i, \quad j = 1, \ldots, p, i = 1, \ldots, N.$$

Note that each $\phi_{2,j}$ is proportional to $\rho_{X_j,\varepsilon}$ and that the direction

$$Z_2 = \sum_{j=1}^{p} \phi_{2,j} X_j = \boldsymbol{\phi}_2^\top \vec{X}$$

places higher weights on the predictors that are most strongly linked to the first residual (which is to say, the component that does not explain Z_1). The process continues in the same way, building directions $Z_3, \ldots, Z_p$ that are strongly linked, in sequence, to the preceding residuals; as the chain starts with the response Y, the directions do take into account both the related response and the predictor structure.[41]

41: The problem of selecting M is tackled as it is in PCA regression.

Summary Due to the **bias-variance trade-off** (see Chapters 19 and 21), we must often strike the right balance in terms of model complexity, which is usually measured in terms of the number of parameters that must be estimated from Tr.

While this allows us to compare completely different models with one another, it also suggests that models that use fewer predictors as inputs are not as complex as those that use the full set of predictors. The full models are not necessarily the ones that perform best (in term of **Te error**), thanks to the **curse of dimensionality**.

Thankfully, predictor subset selection methods can be used to select the best model: while the cross-validation approach is strongly encouraged, other approaches (including shrinkage, feature selection, dimension reduction) could also prove competitive.

23.4.3 Spectral Feature Selection

Text mining tasks often give rise to datasets which are likely to be affected by the CoD; the problem also occurs when dealing with-high resolution images, with each of the millions of pixels it contains viewed as a feature.[42]

42: Such images contain millions of pixels, if not more.

Spectral feature selection attempts to identify "good" or "useful" training features in such datasets by measuring their relevance to a given task *via* spectral analysis of the data.

General Formulation for Feature Selection Let $\mathbf{X} \in \mathbb{R}^{n \times p}$ be a data set with p features and n observations. The problem of $\ell-$feature selection, with $1 \leq \ell \leq p$, can be formulated as an optimization problem [33]:

$$\begin{aligned} &\max_{\mathbf{W}} r(\hat{\mathbf{X}}) \\ \text{s.t. } &\hat{\mathbf{X}} = \mathbf{X}\mathbf{W}, \quad \mathbf{W} \in \{0,1\}^{p \times \ell} \\ &\mathbf{W}^\top \mathbf{1}_{p \times 1} = \mathbf{1}_{\ell \times 1}, \quad \|\mathbf{W}\mathbf{1}_{\ell \times 1}\|_0 = \ell \end{aligned}$$

The **score function** $r(\cdot)$ is the objective which evaluates the relevance of the features in $\hat{\mathbf{X}}$, the data set containing only the features selected by the **selection matrix W** with entries either 0 or 1. To ensure that only the original feature are selected (and not a linear combination of features), the problem stipulates that each column of $\mathbf{W}$ contains only one 1 ($\mathbf{W}^\top \mathbf{1}_{p \times 1} = \mathbf{1}_{\ell \times 1}$); to ensure that exactly ℓ rows contain one 1, the constraint $\|\mathbf{W}\mathbf{1}_{\ell \times 1}\|_0 = l$ is added.

The selected features are often represented by

$$\hat{\mathbf{X}} = \mathbf{X}\mathbf{W} = (f_{i_1}, \ldots, f_{i_\ell}) \quad \text{with } \{i_1, \ldots, i_\ell\} \subset \{1, \ldots, p\}.$$

If $r(\cdot)$ does not evaluate features independently, this optimization problem is NP-hard. To make to problem easier to solve, the features are assumed to be **independent** of one another.[43] In that case, the objective function reduces to

43: Or that their interactions are negligible.

$$\max_{\mathbf{W}} r(\hat{\mathbf{X}}) = \max_{\mathbf{W}} \left(r(f_{i_1}) + \cdots + r(f_{i_l}) \right);$$

the optimization problem can then be solved by selecting the ℓ features with the largest individual scores. The link with **spectral analysis** will now be explored.[44]

44: We have encountered some of these concepts in Section 22.4.2.

Similarity Matrix Let $s_{i,j}$ denote the **pairwise similarity** between observations $\mathbf{x}_i$ and $\mathbf{x}_j$. If class labels $y(\mathbf{x}) \in \{1, \ldots, K\}$ are known for all instances $\mathbf{x}$, the following function can be used

$$s_{i,j} = \begin{cases} \frac{1}{n_k}, & y(\mathbf{x}_i) = y(\mathbf{x}_j) = k \\ 0, & \text{otherwise} \end{cases}$$

where n_k is the number of observations with class label k.

If class labels are not available, a popular similarity function is the **Gaussian radial basis function** (RBF) kernel, given by

$$s_{i,j} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right),$$

where σ is a parameter that is used to control the Gaussian's width.[45] For a given $s_{i,j}$ and n observations, the **similarity matrix** S is an $n \times n$ matrix containing the observations' pairwise similarities, $S(i,j) = s_{i,j}$, $i, j = 1, \ldots, n$.

45: One can think of this as the "reach" of each point.

By convention, $\text{diag}(S) = \mathbf{0}$. Other similarity functions include the following kernels [4]:

1. **linear** – $s_{i,j} = \mathbf{x}_i^\top \mathbf{x}_j + c$, $c \in \mathbb{R}$;
2. **polynomial** – $s_{i,j} = (\alpha \mathbf{x}_i^\top \mathbf{x}_j + c)^d$, $\alpha, c \in \mathbb{R}$,[46] $d \in \mathbb{N}$, and
3. **cosine** – $s_{i,j} = \frac{\mathbf{x}_j^\top \mathbf{x}_i}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|}$, which measures the similarity of 2 vectors by determining the angle between them.[47]

46: For image processing, this kernel is often used with $\alpha = c = 1$.

47: It is often used in high-dimensional applications such as text mining.

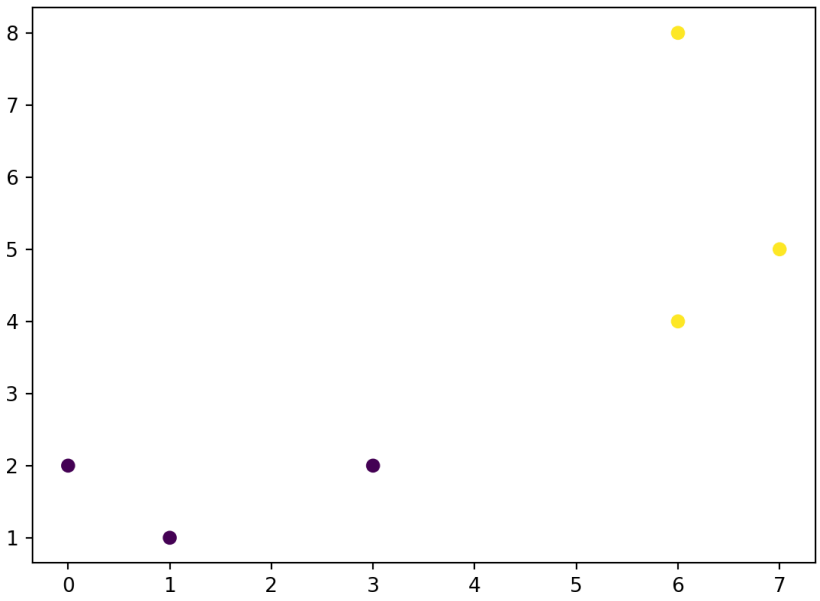
Similarity Graph For each similarity matrix S, one can construct a **weighted graph** $G(V, E)$ in which each observation corresponds to a node and the associated pairwise similarities correspond to the respective edge weights; V is the set of all **vertices** (nodes) and E the set of all graph **edges**. As an example, consider the simple dataset:

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & 3 & 6 & 7 & 6 \\ 1 & 2 & 2 & 4 & 4 & 8 \end{bmatrix}^\top;$$

for validation purposes, we will assume that the first three belong to one group, the last three, to another. The scatter plot is obtained below:

```
from matplotlib import ticker, cm
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import pdist, squareform, cdist
import networkx as nx
from numpy.linalg import eigh,norm
import random # for replicability
```

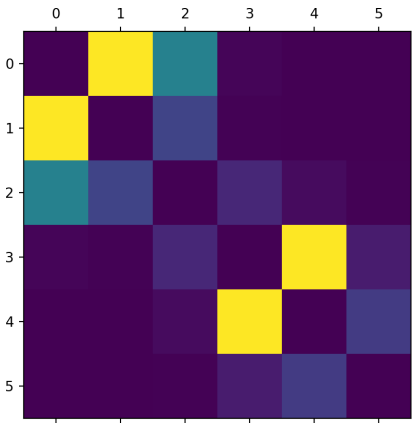
```
# Data
X = np.array([[1,1,1],[0,2,1],[3,2,1],[6,4,1],[7,5,1],[6,8,1]])
Y = np.array([0,0,0,1,1,1])
# Plot
plt.scatter(X[:,0],X[:,1], c=Y)
```



Next, we compute the similarity (adjacency matrix). We use the Gaussian RBF kernel with $\sigma = 0.5$, and `pdist()` from `scipy.spatial.distance` which computes the pairwise distance of each row from an input data matrix. `pdist()` return a vector, which allows a speed boost for the following computation, but it needs to be converted back to a matrix for the next steps.[48]

48: This is done *via* `squareform()`, also from `scipy.spatial.distance`.

```
S = squareform(np.exp(-pdist(X))/((0.5)**2))
plt.matshow(S)
plt.show()
```

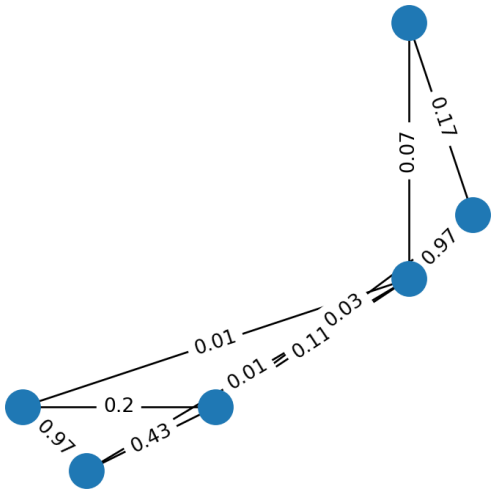


or

$$S = \begin{bmatrix} 0 & 0.972 & 0.428 & 0.012 & 0.003 & 0.001 \\ 0.972 & 0 & 0.199 & 0.007 & 0.002 & 0.001 \\ 0.428 & 0.199 & 0 & 0.109 & 0.027 & 0.005 \\ 0.012 & 0.007 & 0.109 & 0 & 0.972 & 0.073 \\ 0.003 & 0.002 & 0.027 & 0.972 & 0 & 0.169 \\ 0.001 & 0.001 & 0.005 & 0.073 & 0.169 & 0 \end{bmatrix},$$

and the resulting graph G is shown below:

```
G = nx.from_numpy_matrix(S)

# Add the weight (similarity) attribute to the graph
pos = {i : [X[i,0],5*X[i,1]] for i in range(6)}
labels = nx.get_edge_attributes(G,'weight')

# round the similarity (for display)
labels = {k: round(v,2) for k, v in labels.items()}

# Create the edge list and labels. Remove null edges
edge_list = [k for k, v in labels.items() if v != 0]
labels = {k: v for k, v in labels.items() if v != 0}

# Draw it using the edge and labels list, at the right position
nx.draw_networkx_nodes(G,pos)
nx.draw_networkx_edges(G,pos, edgelist=edge_list)
nx.draw_networkx_edge_labels(G,pos, edge_labels=labels)
plt.axis('off')
plt.show()
```



Laplacian Matrix of a Graph The similarity matrix S is also known as the **adjacency matrix** A of the graph G,[49] from which the **degree matrix** D can be constructed:

$$D(i,j) = \begin{cases} d_{i,i} = \sum_{k=1}^{n} a_{i,k}, & i = j \\ 0, & \text{otherwise} \end{cases}$$

49: In certain formulations, the entries of the adjacency matrix A are instead defined to take on the value 1 or 0, depending as to whether the similiarity between the corresponding observations is greater than (or smaller than) some pre-determined threshold τ.

By definition, D is diagonal; the element $d_{i,i}$ can be viewed as an estimate of the **density** around $\mathbf{x}_i$; as $a_{i,k} (= s_{i,k})$ is a measure of similarity between $\mathbf{x}_i$ and $\mathbf{x}_k$, the larger $a_{i,k}$ is, the more similar the two observations are.

A large value of $d_{i,i}$ indicates the presence of one or more observations "near" $\mathbf{x}_i$; conversely, a small value of $d_{i,i}$ suggests that $\mathbf{x}_i$ is isolated.

The **Laplacian** and **normalized Laplacian** matrices are defined as

$$L = D - A \quad \text{and} \quad \mathscr{L} = D^{-1/2} L D^{-1/2},$$

respectively. Since D is diagonal, $D^{-1/2} = \text{diag}\left(d_{i,i}^{-1/2}\right)$.

It can be shown that L and $\mathscr{L}$ are both positive semi-definite matrices. By construction, the smallest eigenvalue of L is 0, with associated eigenvector $\mathbf{1}$, since

$$L\mathbf{1} = (D - A)\mathbf{1} = D\mathbf{1} - A\mathbf{1} \tag{23.1}$$

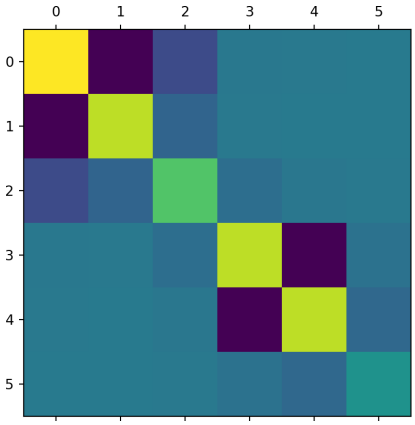
$$= \begin{pmatrix} d_{1,1} \\ \vdots \\ d_{n,n} \end{pmatrix} - \begin{pmatrix} \sum_{k=1}^{n} a_{1,k} \\ \vdots \\ \sum_{k=1}^{n} a_{n,k} \end{pmatrix} = \begin{pmatrix} d_{1,1} - \sum_{k=1}^{n} a_{1,k} \\ \vdots \\ d_{n,n} - \sum_{k=1}^{n} a_{n,k} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} = 0 \cdot \mathbf{1}. \tag{23.2}$$

For $\mathscr{L}$, the corresponding eigenpair is 0 and $\text{diag}(D^{1/2})$ (the proof is similar).

We can compute the degree matrix D and the Laplacian L for the toy example above. For the degree matrix, we use the method `sum()` from `numpy` with the argument `axis=1` to sum over the columns and `diag()` to convert the result into a diagonal matrix.

The Laplacian is simply $L = D - S$.

```
rowsum = S.sum(axis=1)
D = np.diag(rowsum)
L = D - S
plt.matshow(L)
plt.show()
```



Eigenvectors as Cluster Separators The eigenvectors of the Laplacian matrices have some very useful properties relating to features selection. If $\xi \in \mathbb{R}^n$ is an eigenvector of L or $\mathscr{L}$, then ξ can be viewed as a function that assigns a value to each observation in $\mathbf{X}$.

This point-of-view can prove quite useful, as the following simple example from [33] shows. Let $\mathbf{X}$ be constructed of three two-dimensional Gaussians, each with unit variance (and no covariance) but with different means. We start by generating 30 observations for each of the 3 mechanisms, and re-shuffle them into a "random" dataset.

```
random.seed(1234) # for replicability

mean1 = [0,5]
cov1 = [[1,0],[0,1]]
X1 = np.random.multivariate_normal(mean1,cov1,30)
mean2 = [5,0]
cov2 = [[1,0],[0,1]]
X2 = np.random.multivariate_normal(mean2,cov2,30)
mean3 = [-5,-5]
cov3 = [[1,0],[0,1]]
X3 = np.random.multivariate_normal(mean3,cov3,30)

plt.scatter(X1[:,0],X1[:,1])
plt.scatter(X2[:,0],X2[:,1])
plt.scatter(X3[:,0],X3[:,1])

X = np.concatenate((X1,X2,X3), axis=0)
Y = np.array([0] * 30 + [1] * 30 + [2] * 30).reshape((90,1))

df = np.concatenate((X,Y), axis = 1)
np.random.shuffle(df)
X,Y = df[:,:2],df[:,2]
```

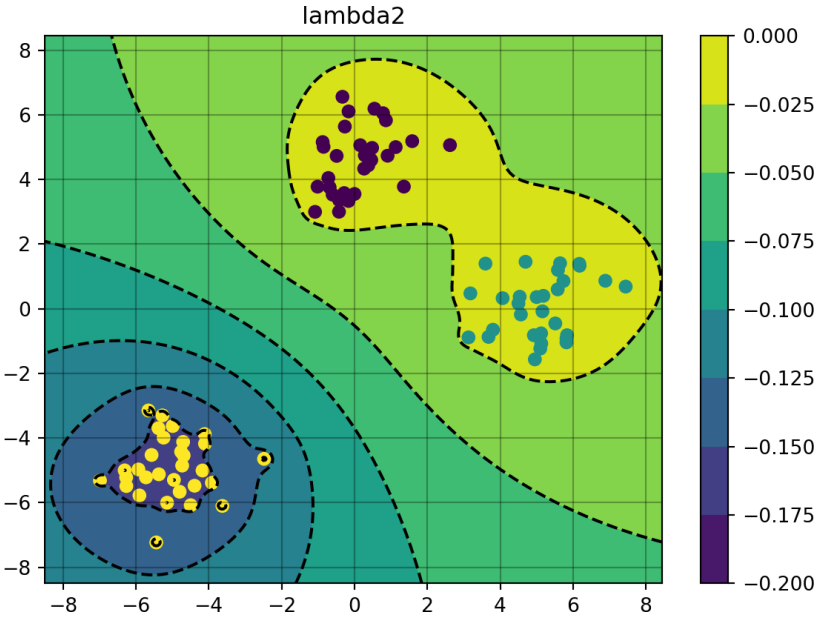
We can then compute the similarity (using Guassian RBF with $\sigma = 1$), adjacency, degree and Laplacian matrix. Using `eigh()` from `numpy` we can find the eigenvalue and eigenvectors of L. This function returns a vector of all the eigenvalues of L and a matrix of all its eigenvectors as columns. According to the convention, the eigenspace is sorted so that the eigenvalues satisfy $0 = \lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \ldots$

```
A = squareform(np.exp(-1 * pdist(X, 'sqeuclidean')))
rowsum = A.sum(axis=1)
D = np.diag(rowsum)
L = D - A

# Find eigenspace of L (vs: eigenvalue vector, es: eigenvector matrix)
vs,es = eigh(L)

# Sort the eigenvalue
arg_sort = vs.argsort()
vs = vs[arg_sort]
es = es[:,arg_sort]
```

We show properties of L's spectrum by providing the **contour plot** of the second eigenvector/eigenvalue pair.[50]

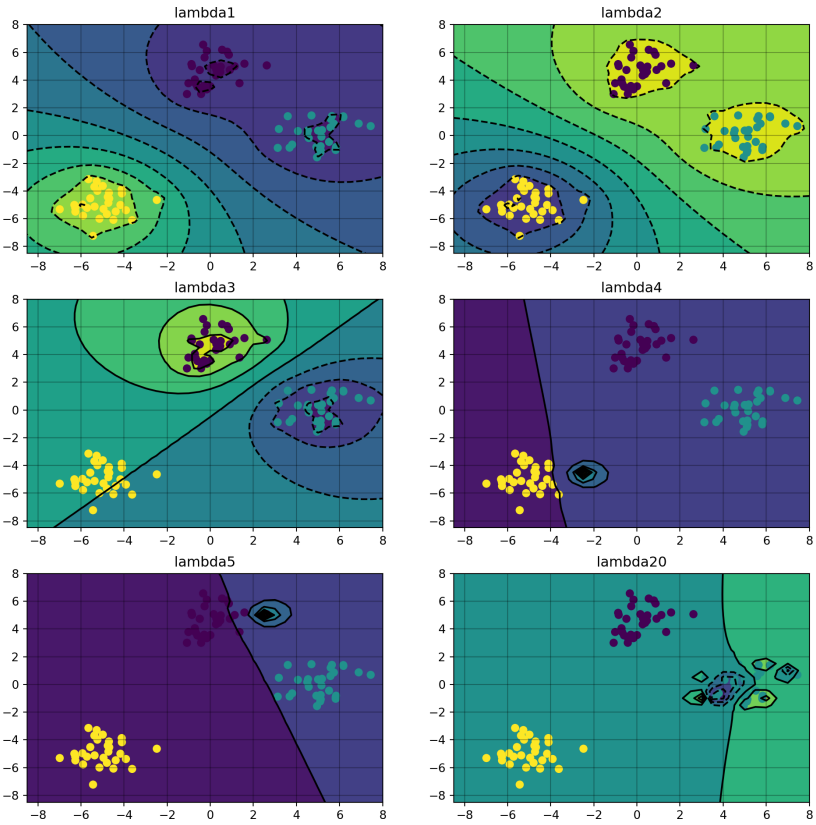
50: Remember, the eigenvectors act as functions in this viewpoint. For a given eigenvector λ_j, the contour value at each point $\mathbf{x}_i$ is the value of the associated eigenvector $\boldsymbol{\xi}_j$ in the i^{th} position, namely $\xi_{j,i}$. For any point $\mathbf{x}$ not in the dataset, the contour value is given by averaging the $\xi_{j,k}$ of the observations $\mathbf{x}_k$ near $\mathbf{x}$, inversely weighted by the distances $\|\mathbf{x}_k - \mathbf{x}\|$.

```
def gen_z(lam = 1):
  """ Compute the meshgrid for z
  """
  for i in range(len(x)):
    for j in range(len(y)):
      dist = cdist([[x_[i,j],y_[i,j]]],X)[0]
      z[i,j] = np.average(es[:,lam], weights= 1/dist)

# Setup the meshgrid
x = np.arange(-8.5,8.5,0.05)
y =  np.arange(-8.5,8.5,0.05)
x_, y_ = np.meshgrid(x,y, indexing='ij')
z = 0*x_

# Index of the eigenvalue to be plotted (0 is the first, etc)
l = 1

# Compute the meshgrid with the gen_z function
gen_z(l)

# Setup and plot
fig,ax = plt.subplots()
cs = ax.contourf(x_, y_, z)
ax.contour(cs,colors='k')
ax.grid(c='k', ls='-', alpha=0.3)
fig.colorbar(cs)
ax.scatter(X[:,0],X[:,1], c=Y)
ax.set_title('lambda{0}'.format(l+1))
plt.show()
```



The next block accomplishes the same tasks, but it does so for potentially more than one eigenvector/eigenvalue pair, and , and aranges the plots in a grid. The contour plot of the ranked eigenvectors $\xi_1, \xi_2, \xi_3, \xi_4, \xi_5$ and ξ_{20}, corresponding to the eigenvalues $\lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \lambda_4 \leq \lambda_5 \leq \lambda_{20}$ are computed and displayed below.

```
x = np.arange(-8.5,8.5,0.5)
y =  np.arange(-8.5,8.5,0.5)
x_, y_ = np.meshgrid(x,y, indexing='ij')
z = 0*x_

# List of index of the eig to be plotted
ls = [0,1,2,3,4,19]
grid_length = 3
grid_width = 2

plt.figure(figsize=(12,12))
for i in range(len(ls)):
  ax = plt.subplot(grid_length,grid_width,i+1)
  z = 0*x_
  gen_z(ls[i])
  cs = ax.contourf(x_, y_, z)
  ax.contour(cs,colors='k')
  ax.grid(c='k', ls='-', alpha=0.3)
  ax.scatter(X[:,0],X[:,1], c=Y)
  ax.set_title('lambda{0}'.format(ls[i]+1))
```



From those plots, it seems as though the first eigenvector does a better job at capturing the cluster structure in the data, while larger values tends to capture more of the sub-cluster structure. One thing to note is that it might appear that λ_1 is as good (or better) as λ_2 and λ_3 to separate the groups, but a closer look at the scale of the contour plot of λ_1 shows that its values have a miniscule range.

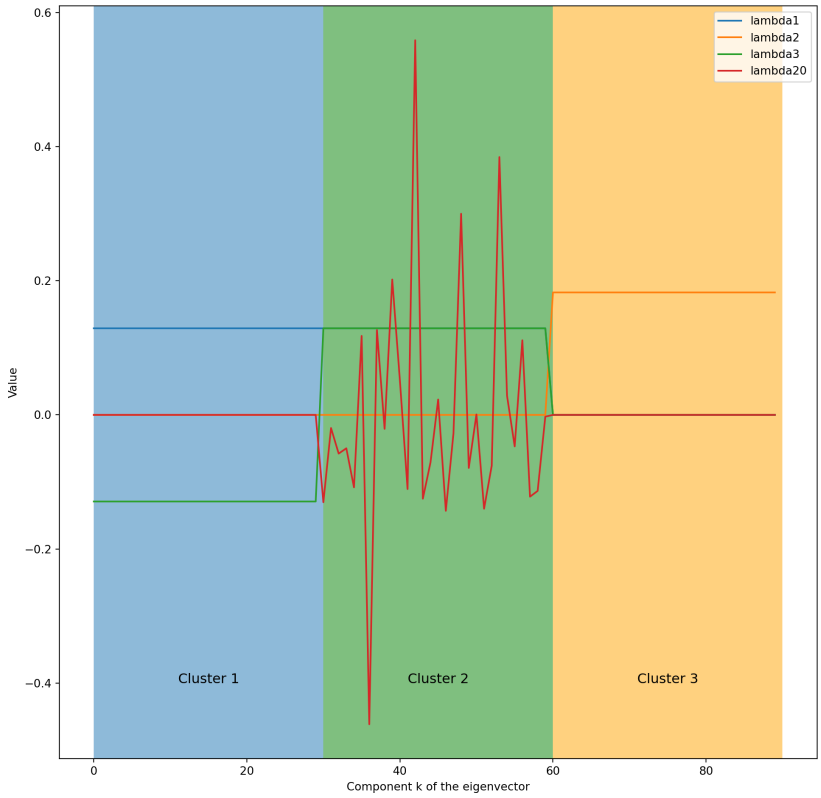
The fact that there is any variation at all is due to floating point errors in the practical computation of the eigenvalue λ_1 and the eigenvector $\boldsymbol{\xi}_1$; as seen previously, these should be exactly 0 and $\mathbf{1}$, respectively.

At any rate, this process shows how the eigenpairs of the Laplacian matrix contains information about the structure of $\mathbf{X}$.

In spectral graph theory, the eigenvalues of the Laplacian measure the **smoothness** of the eigenvectors. An eigenvector is said to be **smooth** if it assigns similar values to points that are near one another.

In the previous example, assume that the data is **unshuffled**, that is, the first k_1 points are in the same cluster, the next k_2 are also in the same, albeit different, cluster, and so on. The next plot shows the smoothness of the eigenvector over each cluster; colour is added to emphasize the cluster limits.

```
# Recompute everything without the shuffling and sorting part
X = np.concatenate((X1,X2,X3), axis=0)
Y = np.array([0] * 30 + [1] * 30 + [2] * 30).reshape((90,1))

A = squareform(np.exp(-1 * pdist(X, 'sqeuclidean')))
rowsum = A.sum(axis=1)
D = np.diag(rowsum)
L = D - A
vs,es = eigh(L)

# List of index of selected eigenvalue.
ls = [0,1,2,19]

# Plot a line for each index in ls
for l in ls:
  plt.plot(es[:,l], label = 'lambda{0}'.format(l+1))

# Color the cluster limits
plt.axvspan(0,30, alpha = 0.5)
plt.text(15,-0.4, 'Cluster 1', fontsize=12)
plt.axvspan(30,60, alpha = 0.5, facecolor='g')
plt.text(45,-0.4, 'Cluster 2', fontsize=12)
plt.axvspan(60,90, alpha = 0.5, facecolor='orange')
plt.text(75,-0.4, 'Cluster 3', fontsize=12)

plt.legend()
plt.ylabel('Value')
plt.xlabel('Component k of the eigenvector')
plt.show()
```



Both λ_2 and λ_3 are fairly smooth, as they seem to be piece-wise constant on each cluster, whereas λ_{20} is all over the place on cluster 1 and constant on the rest of the data. As discussed previously λ_1 is constant over the entirety of the dataset, marking it as maximally smooth but not very useful from the perspective of differentiating data structure.[51]

51: As a reminder, the eigenvalues themselves are ordered in increasing sequence: for the current example,

$$\lambda_1 = 0 \leq \lambda_2 = 1.30 \times 10^{-2} \leq \lambda_3 = 3.94 \times 10^{-2} \leq \cdots \lambda_{20} = 2.95 \leq \cdots$$

Indeed, let $\mathbf{x} \in \mathbb{R}^n$. then

$$\begin{aligned}
\mathbf{x}^\top L\mathbf{x} = \mathbf{x}^\top D\mathbf{x} - \mathbf{x}^\top A\mathbf{x} &= \sum_{i=1}^{n} d_i x_i^2 - \sum_{i,j=1}^{n} a_{i,j} x_i x_j \\
&= \frac{1}{2}\left(\sum_{i=1}^{n} d_i x_i^2 - 2\sum_{i,j=1}^{n} a_{ij} x_i x_j + \sum_{j=1}^{n} d_j x_j^2\right) \\
&= \frac{1}{2}\sum_{i,j=1}^{n} a_{ij}(x_i - x_j)^2
\end{aligned}$$

If $\mathbf{x} = \boldsymbol{\xi}$ is a normalized eigenvector of L, then $\boldsymbol{\xi}^\top L\boldsymbol{\xi} = \lambda \boldsymbol{\xi}^\top \boldsymbol{\xi} = \lambda$, thus

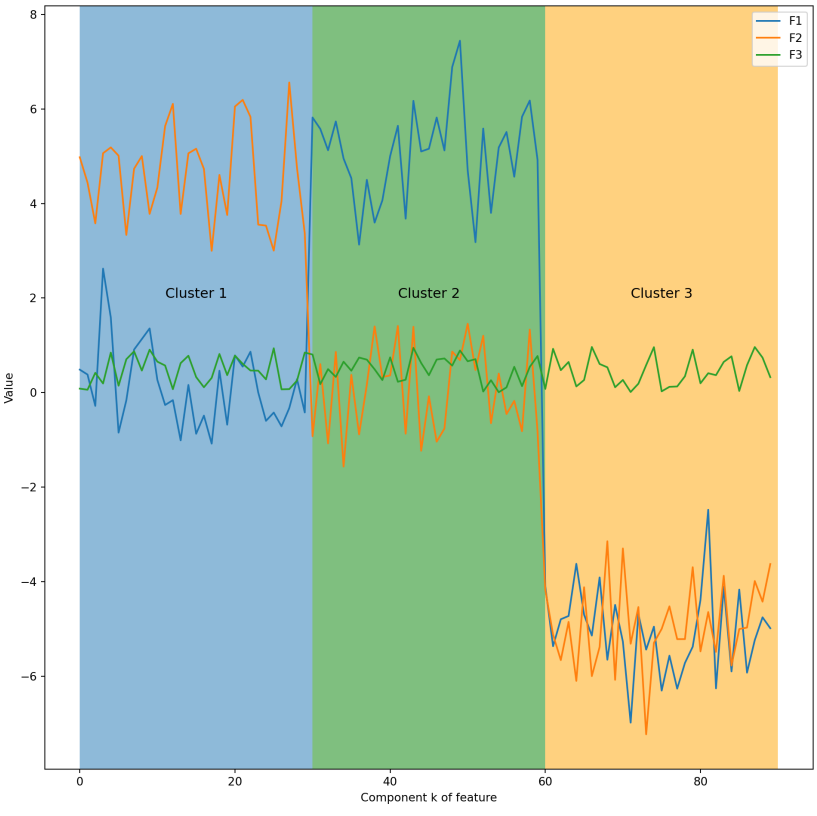
$$\lambda = \boldsymbol{\xi}^\top L\boldsymbol{\xi} = \frac{1}{2}\sum_{i,j=1}^{n} a_{ij}(\xi_i - \xi_j)^2.$$

Instinctively, if the eigenvector component does not vary a lot for observations that are near one another, one would expect the corresponding eigenvalue to be small; this result illustrates why the small magnitude of the eigenvalue is a good measure of the smoothness of its associated eigenvector.

Feature Ranking We can use the above discussion as a basis for feature selection. If $\mathbf{x}$ is not en eigenvector of L, the value $\mathbf{x}^\top L\mathbf{x}$ can also be seen as a measure of how much $\mathbf{x}$ varies locally. This can be used to measure how meaningful a feature $\mathbf{f} \in \mathbb{R}^n$ is.

In the current example, the only two features are the Euclidean coordinates of the observations: $\mathbf{f}_1$ and $\mathbf{f}_2$. We also add a **useless** feature $\mathbf{f}_3$ to the dataset (distributed uniformly across the clusters).

```
# Add a random feature
U1 = np.random.uniform(size=(90,1))
X = np.concatenate((X,U1), axis=1)
f = [0,1,2]

for i in f:
  plt.plot(X[:,i], label = 'F{0}'.format(i+1))

plt.axvspan(0,30, alpha = 0.5)
plt.text(15,2, 'Cluster 1', fontsize=12)
plt.axvspan(30,60, alpha = 0.5, facecolor='g')
plt.text(45,2, 'Cluster 2', fontsize=12)
plt.axvspan(60,90, alpha = 0.5, facecolor='orange')
plt.text(75,2, 'Cluster 3', fontsize=12)

plt.legend()
plt.ylabel('Value')
plt.xlabel('Component k of feature')
plt.show()
```



The plot shows that the third feature is not able to distinguish between the clusters. However, we also have:

```
for i in [0,1,2]:
  f = X[:,i]
  print("F{0}".format(i+1), "->",f.T.dot(f.dot(L)))
```

```
F1 -> 105.35092482283625
F2 -> 110.6011478717457
F3 -> 46.90787692252817
```

Thus

$$\mathbf{f}_1^\top L\mathbf{f}_1 = 94.3, \quad \mathbf{f}_2^\top L\mathbf{f}_2 = 102.5, \quad \mathbf{f}_3^\top L\mathbf{f}_3 = 41.7;$$

by the previous assumption relating the magnitude of $\boldsymbol{\xi}^\top L\boldsymbol{\xi}$ to the smoothness of $\boldsymbol{\xi}$, this would seem to indicate that $\mathbf{f}_3$ is a "better" feature than the other two.

The problem is that the value of $\mathbf{f}_i^\top L\mathbf{f}_i$ is affected by the respective norms of $\mathbf{f}_i$ and L. This need to be addressed.

The relation between L and $\mathscr{L}$ yields

$$\mathbf{f}_i^\top L\mathbf{f}_i = \mathbf{f}_i^\top D^{1/2}\mathscr{L}D^{1/2}\mathbf{f}_i = (D^{1/2}\mathbf{f}_i)^T\mathscr{L}(D^{1/2}\mathbf{f}_i).$$

Set $\tilde{\mathbf{f}}_i = (D^{1/2}\mathbf{f}_i)$ and $\hat{\mathbf{f}}_i = \tilde{\mathbf{f}}_i/\|\tilde{\mathbf{f}}_i\|$. The **feature score metric** φ_1 is a normalized version of the smoothness measure:

$$\varphi_1(\mathbf{f}_i) = \hat{\mathbf{f}}_i^\top\mathscr{L}\hat{\mathbf{f}}_i, \quad i = 1, \ldots, p.$$

For φ_1, smaller values are better. The scoring function can also be defined using the spectral decomposition of $\mathscr{L}$.

Suppose that $(\lambda_k, \boldsymbol{\xi}_k)$, $1 \le k \le n$ are **eigenpairs** of $\mathscr{L}$ and let $\alpha_k = \hat{\mathbf{f}}_i^\top\boldsymbol{\xi}_k$, for a given i. Then

$$\varphi_1(\mathbf{f}_i) = \sum_{k=1}^{n} \alpha_k^2\lambda_k, \quad \text{with } \sum_{k=1}^{n} \alpha_k^2 = 1.$$

Indeed, let $\mathscr{L} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^\top$ be the eigen-decomposition of $\mathscr{L}$. By construction, $\mathbf{U} = [\xi_1|\xi_2|\cdots|\xi_n]$ and $\boldsymbol{\Sigma} = \text{diag}(\lambda_k)$, so that

$$\begin{aligned}\varphi_1(\mathbf{f}_i) &= \hat{\mathbf{f}}_i^\top\mathscr{L}\hat{\mathbf{f}}_i = \hat{\mathbf{f}}_i^\top\mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^\top\hat{\mathbf{f}}_i \\ &= (\alpha_1, \ldots, \alpha_n)\boldsymbol{\Sigma}(\alpha_1, \ldots, \alpha_n)^\top = \sum_{k=1}^{n} \alpha_k^2\lambda_k.\end{aligned}$$

This representation allows for a better comprehension of the φ_1 score; α_k is the cosine of the angle between the normalized feature $\hat{\mathbf{f}}_i$ and eigenvector $\boldsymbol{\xi}_k$. If a feature aligns with "good" eigenvectors (i.e., those with small eigenvalues), its φ_1 score will also be small.

The larger α_1^2 is, the smaller $\sum_{k=2}^{n} \alpha_k^2$ is; this is problematic because, in such cases, a small value of φ_1 indicates smoothness but not separability.

To overcome this issue, φ_1 can be normalized by $\sum_{k=2}^{n} \alpha_k^2$, which yields a new scoring function:

$$\varphi_2(\mathbf{f}_i) = \frac{\sum_{k=1}^{n} \alpha_k^2 \lambda_k}{\sum_{k=2}^{n} \alpha_k^2} = \frac{\hat{\mathbf{f}}_i^\top \mathcal{L} \hat{\mathbf{f}}_i}{1 - \left(\hat{\mathbf{f}}_i^\top \boldsymbol{\xi}_1\right)}$$

A small value for φ_2 once again indicates that a feature closely aligns with "good" eigenvectors.

Another ranking feature is closely related to the other two. According to spectral clustering, the first k non-trivial eigenvectors form an optimal set of **soft cluster indicators** that separate the graph G into k connected parts. Therefore, we define φ_3 as

$$\varphi_3(\mathbf{f}_i, k) = \sum_{j=2}^{k} (2 - \lambda_j) \alpha_j^2.$$

Contrary to the other scoring functions, φ_3 assigns larger value to feature that are more relevant. It also prioritizes the leading eigenvectors, which helps to reduce noise. Using this ranking function requires a number of categories or clusters k to be selected (depending on the nature of the ultimate task at hand); if this value is unknown, k becomes a hyper-parameter to be tuned.

The feature score metrics are implemented as below:

```
D_sq = np.sqrt(D)
L_norm = D_sq.dot(L).dot(D_sq)

def score_1(i):
  f_tilde = D_sq.dot(X[:,i])
  f_hat = f_tilde / norm(f_tilde)
  return f_hat.dot(L_norm).dot(f_hat)

def score_2(i):
  f_tilde = D_sq.dot(X[:,i])
  f_hat = f_tilde / norm(f_tilde)
  phi_1 = f_hat.dot(L_norm).dot(f_hat)
  return phi_1 / (1 - (f_hat.dot(es[:,0])**2))

def score_3(i,k):
  f_tilde = D_sq.dot(X[:,i])
  f_hat = f_tilde / norm(f_tilde)
  alpha = f_hat.dot(es)
  temp = (2 - vs[1:k]) * (alpha[1:k])**2
  return np.sum(temp)

from tabulate import tabulate

n_feature = X.shape[1]
results = {'phi_1':[], 'phi_2':[], 'phi_3': []}
k = 3
```

```
for i in range(n_feature):
  results['phi_1'].append(score_1(i))
  results['phi_2'].append(score_2(i))
  results['phi_3'].append(score_3(i,k))

print(tabulate(results,
                      headers="keys",
                      showindex=True,
                      tablefmt="simple",
                      numalign="left"))
```

```
    phi_1    phi_2    phi_3
--  -------  -------  --------
0   2.5516   3.28365  1.37706
1   2.71268  3.47861  1.40086
2   17.0148  31.8369  0.433994
```

This make more sense, as the pattern is similar to the pattern obtained for the eigenvalues: $\mathbf{f}_1, \mathbf{f}_2$, being able to differentiate the clusters, have smaller φ_1 scores than $\mathbf{f}_3$. Returning to the current example, while the score of the useless feature 3, $\varphi_1(\mathbf{f}_3)$ is larger than the other scores, it is still small when compared to the eigenvalues of L. This is due to the fact the $\mathbf{f}_3$ and ξ_1 are nearly co-linear.

Computing φ_2 for our three features yields a larger distinction between the real features and the random, useless one than with φ_1.

Regularization There is one glaring problem with the ranking functions that have been defined previously: they all assume the existence of a gap between subsets of "large" and "small" eigenvalues. For clearly separated data, that is to be expected; but in noisy data, this gap may be negligible, which leads to an increase in the score value of poor features [32].

This issue can be tackled by applying a **spectral matrix function** $\gamma(\cdot)$ to the Laplacian $\mathcal{L}$, replacing the original eigenvalues by **regularized eigenvalues** as follows:

$$\gamma(\mathcal{L}) = \sum_{j=1}^{n} \gamma(\lambda_j)\xi_j\xi_j^\top.$$

In order for this to work properly, γ needs to be (strictly) increasing. Examples of such regularization functions include:

$\gamma(\lambda)$	**(name)**
$1 + \sigma^2\lambda$	(regularized Laplacian)
$\exp(\sigma^2/2\lambda)$	(diffusion Process)
$\lambda^\nu,\ \nu \geq 2$	(high-order polynomial)
$(a - \lambda)^{-p},\ a \geq 2$	(p-step random walk)
$(\cos \lambda\pi/4)^{-1}$	(inverse cosine)

The ranking function $\varphi_1, \varphi_2, \varphi_3$ can be regularized *via*

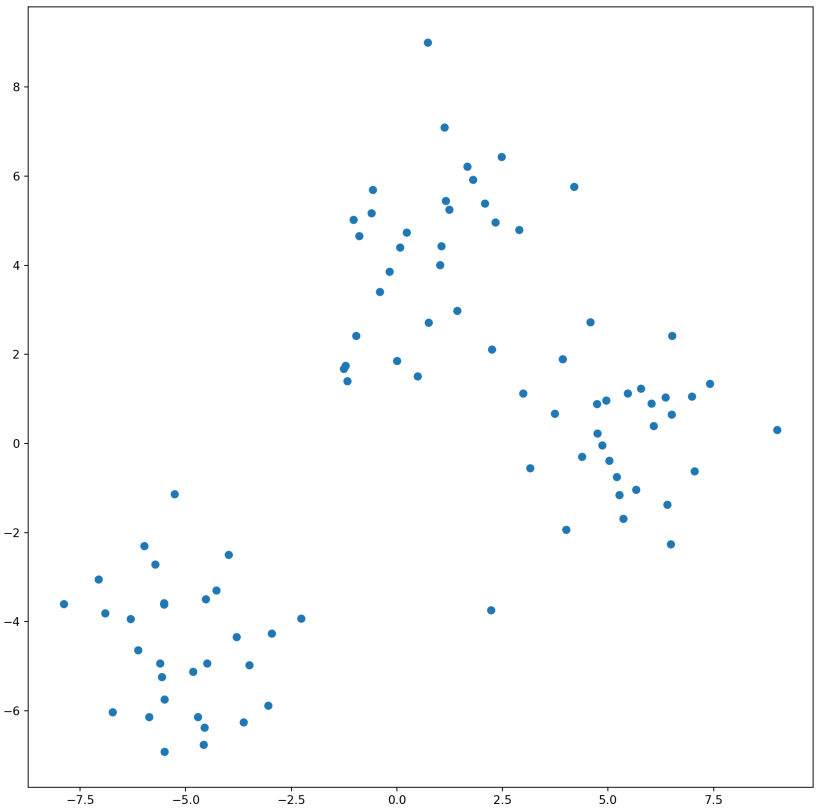
$$\hat{\varphi}_1(\mathbf{f}_i) = \sum_{k=1}^{n} \alpha_k^2 \gamma(\lambda_k)$$

$$\hat{\varphi}_2(\mathbf{f}_i) = \frac{\hat{\mathbf{f}}_i^\top \gamma(\mathcal{L}) \hat{\mathbf{f}}_i}{1 - \left(\hat{\mathbf{f}}_i^\top \xi_1\right)}$$

$$\hat{\varphi}_3(\mathbf{f}_i) = \sum_{j=2}^{k} (\gamma(2) - \gamma(\lambda_j)) \alpha_j^2$$

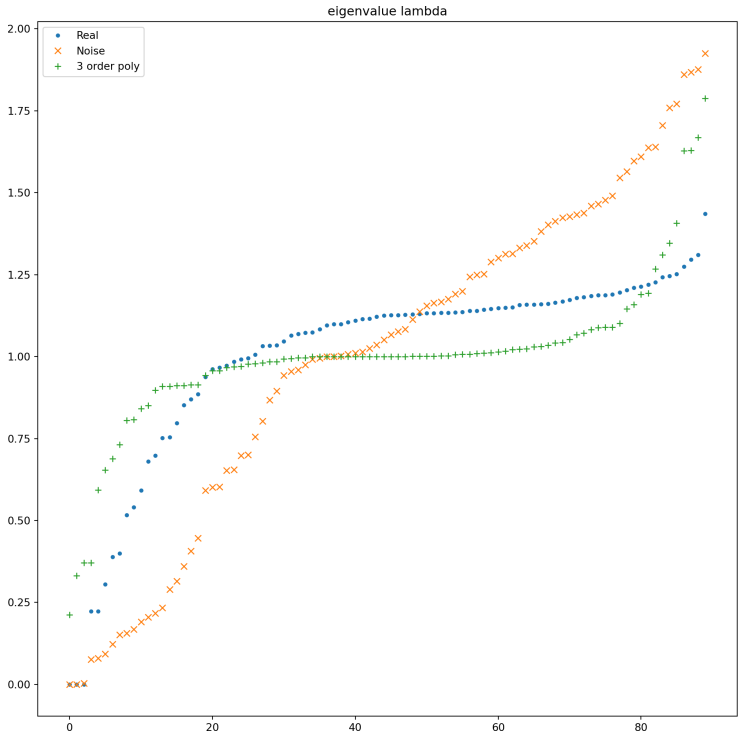
To illustrate how this regularization process can help reduce noise (still using the framework from the previous example), $\mathbf{X}$ was contaminated with random values from a normal distribution with a variance of 1.5.

```
random.seed(5678) # for replicability
noise = np.random.normal(0, 1.3, 90*3).reshape(90,3)
X_noise = X + noise
plt.scatter(X_noise[:,0], X_noise[:,1])
plt.show()
```



We plot the components of the eigenvalues of three normalized Laplacians: one from the original data, one from the noisy data, and one from the noisy data with a 3rd order polynomial regularization.

```
def isqrt(x):
  if x == 0:
    return 0
  else:
    return x**(-0.5)

v_isqrt = np.vectorize(isqrt)

def find_eig(X, k = lambda x:x):
  A = squareform(np.exp(-1 * pdist(X, 'sqeuclidean')))
  rowsum = A.sum(axis=1)
  D = np.diag(rowsum)
  L = D - A
  D_is = v_isqrt(D)
  L_norm = k(D_is.dot(L).dot(D_is))
  vs,es = eigh(L_norm)
  arg_sort = vs.argsort()
  vs = vs[arg_sort]
  es = es[:,arg_sort]
  return vs,es, L_norm

vs,es,L = find_eig(X)
vs_n,es_n,L_noise = find_eig(X_noise)
vs_k,es_k, L_k = find_eig(X_noise, k = lambda x:x**3)

plt.plot(vs, '.', label = 'Real')
plt.plot(vs_n, 'x', label = 'Noise')
plt.plot(vs_k, "+", label = '3 order poly')
plt.title('eigenvalue lambda')
plt.legend()
```

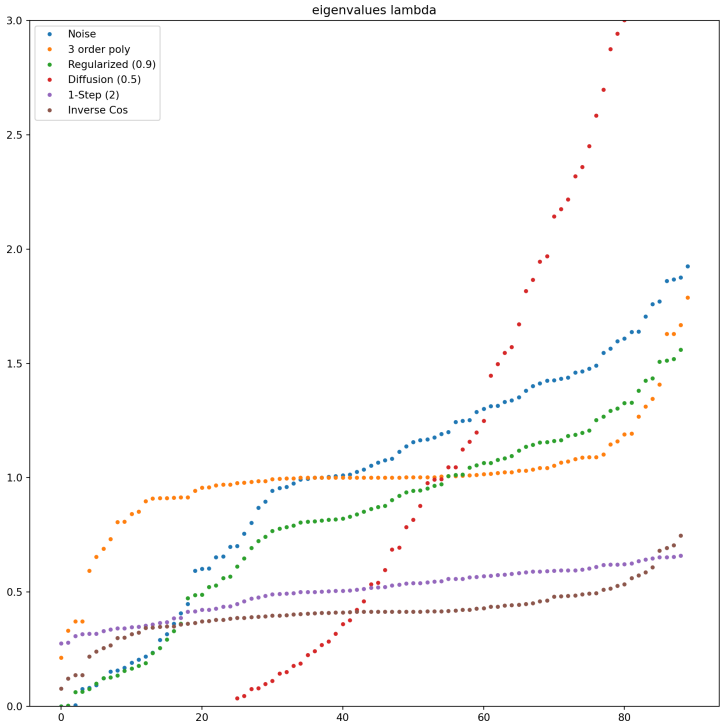


The preceding plot shows the effect of noise on $\mathscr{L}$'s: it tends to linearize the eigenvalues, and this provides much support to the poorer eigenvectors. The eigenvalues of the noisy Laplacian have been regularized using the standard cubic $\gamma(\lambda) = \lambda^3$; the distinction between the first eigenvalues and the rest is clear.

We can compare different kernels:

- **regularized Laplacian** – $\gamma(\lambda) = 1 + (0.9)^2\lambda$
- **high-order polynomial** – $\gamma(\lambda) = \lambda^3$
- **diffusion process** – $\gamma(\lambda) = \exp((0.3)^2/2\lambda)$
- p**–step random walk** – $\gamma(\lambda) = (2 - \lambda)^{-1}$
- **inverse cosine** – $\gamma(\lambda) = \cos(\lambda\pi/4)^{-1}$

```
vs_n,es_n,_ = find_eig(X_noise)
vs_reg, es_reg,_ = find_eig(X_noise, k = lambda x:1 + (0.9)**2 * x)
vs_k3,es_k3,_ = find_eig(X_noise, k = lambda x:x**3)
vs_diff, es_diff,_ = find_eig(X_noise, k = lambda x: np.exp((0.3)**2/(2*x)))
vs_1step, es_1step,_ = find_eig(X_noise, k = lambda x: (2-x)**(-1))
vs_cos, es_cos, _ = find_eig(X_noise, k = lambda x: np.cos(x * (3.1416/4))**(-1))

plt.plot(vs_n, '.', label = 'Noise')
plt.plot(vs_k, '.', label = '3 order poly')
plt.plot(vs_reg, '.', label = 'Regularized (0.9)')
plt.plot(vs_diff, '.', label = 'Diffusion (0.5)')
plt.plot(vs_1step, '.', label = '1-Step (2)')
plt.plot(vs_cos, '.', label = 'Inverse Cos')

plt.ylim(0,3)
plt.title('eigenvalues lambda')
plt.legend()
plt.show()
```



The choice of a specific regularization function depends on the context and the goals of the data analysis task; for large datasets, considerations of ease of computation may also form part of the selection strategy.

Spectral Feature Selection with SPEC The remarks from the previous subsections can be combined to create a feature selection framework called **SPEC** [22]:

1. using a specified similarity function s, construct a similarity matrix S of the data $\mathbf{X}$ (optionally with labels $\mathbf{Y}$);
2. construct the graph G of the data;
3. extract the normalized Laplacian $\mathscr{L}$ from this graph;
4. compute the eigenpairs (eigenvalues and eigenvectors) of $\mathscr{L}$;
5. select a regularization function $\gamma(\cdot)$;
6. for each feature $\mathbf{f}_i$, $i = 1, \ldots, p$, compute the relevance $\hat{\varphi}(\mathbf{f}_i)$, where $\hat{\varphi} \in \{\hat{\varphi}_1, \hat{\varphi}_2, \hat{\varphi}_3\}$, and
7. return the features in descending order of relevance.

In order for SPEC to provide "good" results, proper choices for the similarity, ranking, and regularization functions are needed. Among other considerations, the similarity matrix should reflect the true relationships between the observations.

Furthermore, if the data is **noisy**, it might be helpful to opt for $\hat{\varphi} = \hat{\varphi}_3$ and/or $\gamma(\lambda) = \lambda^\nu$, $\nu \geq 2$. When the gap between the small and the large eigenvalues is wide, $\hat{\varphi} = \hat{\varphi}_2$ or $\hat{\varphi} = \hat{\varphi}_3$ usually provide **good choices**, although $\hat{\varphi}_2$ has been shown to be more robust [33].

23.4.4 Uniform Manifold Approximation and Projection

The feature selection and dimension reduction landscape is in flux, and there are more recent (and sophisticated) developments that are generating a lot of interest. Case in point, consider **Uniform Manifold Approximation and Projection** (UMAP) methods.

Dimensionality Reduction and UMAP A mountain is a 3-dimensional object.[52] And the surface of a mountain range is 2-dimensional – it can be represented with a flat map – even though the surface, and the map for that matter, still exist in 3-dimensional space.[53]

What does it mean to say that a shape is q-dimensional for some q? What is a **shape**, even?

Shapes could be lines, cubes, spheres, polyhedrons, or more complicated things. In geometry, the customary way to represent a shape is *via* a set of points $S \subseteq \mathbb{R}^p$.

A circle is the set of points whose distance to a fixed point (the centre) is **exactly equal** to the radius r, say, whereas a disk is the set of points whose distance to the centre is **at most** the radius.

In the mountain example, $p = 3$ for both the mountains S_m and the mountain surface S_s. So the question is, when is the **(effective) dimension** of a set S less than p, and how is that dimension calculated?

52: When we consider the world at a low resolution, at least.

53: In the parlance of the field, we say that the surface is **embedded** in $\mathbb{R}^3$.

It turns out that there are multiple definitions of the **dimension** q of a set $S \subseteq \mathbb{R}^p$ [6]:

- the smallest q for which S is q-dimensional manifold;
- how many nontrivial n^{th} homology groups of S there are;
- how the "size" of the set scales as the coordinates scale.

A **q-dimensional manifold** is a set where each small region is approximately the same as a small region of $\mathbb{R}^q$. For instance, if a small piece of a (stretchable) sphere is cut out with a cookie cutter, it could theoretically be bent so that it looks like it came from a flat plane, without changing its "essential" shape.

Dimensionality reduction is more than just a matter of selecting a definition and computing q, however. Indeed, any dataset $\mathbf{X}$ is necessarily finite and is thus, by definition, actually 0−dimensional; the object of interest is the shape that the data **would** form if there were infinitely many available data points, or, in other words, the **support of the distribution** generating the data.

Furthermore, any dataset is probably noisy and may only **approximately lie** in a lower-dimensional shape.

Lastly, it is not clear how to build an algorithm that would, for example, determine what all the **homology groups** of some set S are. The problem is quite thorny. Let $X \subseteq \mathbb{R}^p$ be a finite set of points. A **dimensionality reducer** for $\mathbf{X}$ is a function $f_{\mathbf{X}} : \mathbf{X} \to \mathbb{R}^q$, where $q < p$, which satisfies certain properties that imply that $f_{\mathbf{X}}(\mathbf{X})$ has similar structure to $\mathbf{X}$.[54]

54: In the remainder of this section, the subscript is dropped. Note that q is assumed, not found by the process.

Various dimensionality reducers were discussed in Section 23.2; they each differ based on the relationship between $\mathbf{X}$ and $f_{\mathbf{X}}(\mathbf{X})$.

For instance, in PCA, the dataset $\mathbf{X}$ is first translated so that its points (or at least its "principal components") lie in a linear subspace. Then q unit-length linear basis elements are chosen to span a subspace, projection onto which yields an affine map f from $\mathbf{X}$ to $\mathbb{R}^q$ that preserves Euclidean distances between points (a rigid transformation), assuming that the non-principal dimensions are ignored.

PCA seems reasonable but what if a rigid transformation down to $\mathbb{R}^q$ is not possible? As an example, consider the **swiss roll** of Figure 23.13, which is a loosely rolled up rectangle in 3-dimensional space. What can be preserved when we "reduce" this space? Only the local structure? The global structure?

UMAP is a dimension reduction method that attempts to approximately preserve **both the local and global structure**. It can be especially useful for visualization purposes, i.e., reducing to $q = 3$ or fewer dimensions. While the semantics of UMAP can be stated in terms of graph layouts, the method was derived from abstract topological assumptions. For a full treatment and mathematical properties, see [15].

Note that UMAP works best when the data $\mathbf{X}$ is evenly distributed on its support $\mathcal{S}$. In this way, the points of $\mathbf{X}$ "cover" $\mathcal{S}$ and UMAP can determine where the true gaps or holes in S are.

UMAP Semantics Let the (scaled) data be denoted by $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, where $\mathbf{x}_i \in \mathbb{R}^p$ for all i; let $d : \mathbf{X} \times \mathbf{X} \longrightarrow \mathbb{R}_{\geq 0}$ be a distance function, and let $k \geq 1$ be an integer.

Consider a **directed graph** graph $D = (V, E, A)$, with

- vertices $V(D) = \mathbf{X}$;
- edges $E(D)$ consisting of the ordered pairs $(\mathbf{x}_i, \mathbf{x}_j)$ such that $\mathbf{x}_j$ is one of the k nearest neighbours of $\mathbf{x}_i$ according to d;
- weight function $W : E(D) \longrightarrow \mathbb{R}$ such that

$$w(\mathbf{x}_i, \mathbf{x}_{i,j}) = \exp\left(\frac{-\max(0, d(\mathbf{x}_i, \mathbf{x}_{i,j}) - \rho_i)}{\sigma_i}\right),$$

  where $\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,k}$ are the k nearest neighbours of $\mathbf{x}_i$ according to d, ρ_i is the minimum nonzero distance from $\mathbf{x}_i$ to any of its neighbours, and σ_i is the unique real solution of

$$\sum_{j=1}^{k} \exp\left(\frac{-\max(0, d(x_i, x_{i,j}) - \rho_i)}{\sigma_i}\right) = \log_2(k),$$

  and
- A is the weighted adjacency matrix of D with vertex ordering $\mathbf{x}_1, \ldots, \mathbf{x}_n$.

Define a symmetric matrix

$$B = A + A^\top - A \circ A^\top,$$

where $\circ$ is **Hadamard's component-wise product**.

The graph $G = (V, W, B)$ has the same vertex set, the same vertex ordering, and the same edge set as D, but its edge weights are given by B. Since B is symmetric, G can be considered to be undirected.

UMAP returns the (reduced) points $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n) \in \mathbb{R}^q$ by finding the position of each vertex in a **force directed graph layout**, which is defined *via* a graph, an attractive force function defined on edges, and a repulsive force function defined on all pairs of vertices.

Both force functions produce **force values** with a direction and magnitude based on the pair of vertices and their respective positions in $\mathbb{R}^q$.

To compute the **layout**, initial positions in $\mathbb{R}^q$ are chosen for each vertex, and an iterative process of translating points based on their attractive and repulsive forces is carried out until a convergence criterion is met. In UMAP, the attractive force between vertices $\mathbf{x}_i, \mathbf{x}_j$ at positions $y_i, y_j \in \mathbb{R}^q$, respectively, is

$$\frac{-2ab\|y_i - y_j\|_2^{2(b-1)}}{1 + \|y_i - y_j\|_2^2} w(x_i, x_j)(y_i - y_j),$$

where a and b are parameters, and the repulsive force is

$$\frac{b(1 - w(x_i, x_j))(y_i - y_j)}{(0.001 + \|y_i - y_j\|_2^2)(1 + \|y_i - y_j\|_2^2)}.$$

There are a number of important **free parameters** to select, namely

- k: nearest neighbor neighborhood count;
- q: target dimension;
- d: distance function, e.g. Euclidean metric.

The UMAP documentation states,

> low values of k will force UMAP to concentrate on very local structure (potentially to the detriment of the big picture), while large values will push UMAP to look at larger neighborhoods of each point . . . losing fine detail structure for the sake of getting the broader structure of the data [15].

The user may set these parameters to appropriate values for the dataset. The choice of a distance metric plays the same role as in clustering, where closer pairs of points are considered to be more similar than farther pairs. There is also a minimum distance value used within the force directed layout algorithm which says how close together the positions may be.

Example We compare various dimensionality reducers for a number of datasets (adapted from the UMAP documentation).[55] Let us start by installing the required `Python` modules.

55: Careful: the correct `Python` package to install is `umap-learn`, not `umap`.

```
import warningswarnings.filterwarnings('ignore')
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, decomposition, manifold, preprocessing
from colorsys import hsv_to_rgb
import umap.umap_ as umap
```

We will plot the two-dimensional reduction of five different datasets; the points will be coloured to get a sense of which points got sent where (otherwise we would only know the shape of the reduced dataset, but have no way to tell how the local structure is affected by the reducers).

We use $t-$SNE, Isomap, MDS (multidimensional scaling), PCA, and UMAP. The 5 datasets are:

- a set consisting of 4 distinct 10-dimensional Gaussian distributions;
- the digit classification dataset;
- the wine characteristics dataset (essentially 1D);
- a 2D rectangle rolled up in 3D space (colours indicate the position along the unrolled rectangle), and
- points on the 2D surface of a 3D sphere (which is not homeomorphic to $\mathbb{R}^2$)); even with the north pole removed, the stereographic projection would map points close to the north pole arbitrarily far from the origin (colour hue indicates the angle around the equator and darkness indicates distance from south pole).

```
blobs, blob_labels = datasets.make_blobs(
    n_samples=500, n_features=10, centers=4
  )
```

```
digits = datasets.load_digits(n_class=10)

wine = datasets.load_wine()

swissroll, swissroll_labels = datasets.make_swiss_roll(
    n_samples=1000, noise=0.1
  )

sphere = np.random.normal(size=(600, 3))

# scale points to have same distance from origin
sphere = preprocessing.normalize(sphere)

# compute colours in HSV format
sphere_hsv = np.array([
        (
          (np.arctan2(c[1], c[0]) + np.pi) / (2 * np.pi),
          np.abs(c[2]), min((c[2] + 1.1), 1.0),
        )
      for c in sphere
    ])

# convert colours to RGB format
sphere_colors = np.array([hsv_to_rgb(*c) for c in sphere_hsv])
```

Next we set parameters for the reducer algorithms. For UMAP, we set `min_dist` to 0.3 which will spread out the points to a noticable degree. We also set the number k of nearest neighbours to 30.

The choices for the other reduces are shown in the code block.

In practice, we would typically set these parameters on a dataset-by-dataset basis, but for illustration purposes, we do not need to fine tune the choices.

```
reducers = [
    (manifold.TSNE, {"perplexity": 50}),
    (manifold.Isomap, {"n_neighbors": 30}),
    (manifold.MDS, {}),
    (decomposition.PCA, {}),
    (umap.UMAP, {"n_neighbors": 30, "min_dist": 0.3}),
]

test_data = [
    (blobs, blob_labels),
    (digits.data, digits.target),
    (wine.data, wine.target),
    (swissroll, swissroll_labels),
    (sphere, sphere_colors),
]

dataset_names = ["Blobs", "Digits", "Wine", "Swiss Roll",
    "Sphere"]
```

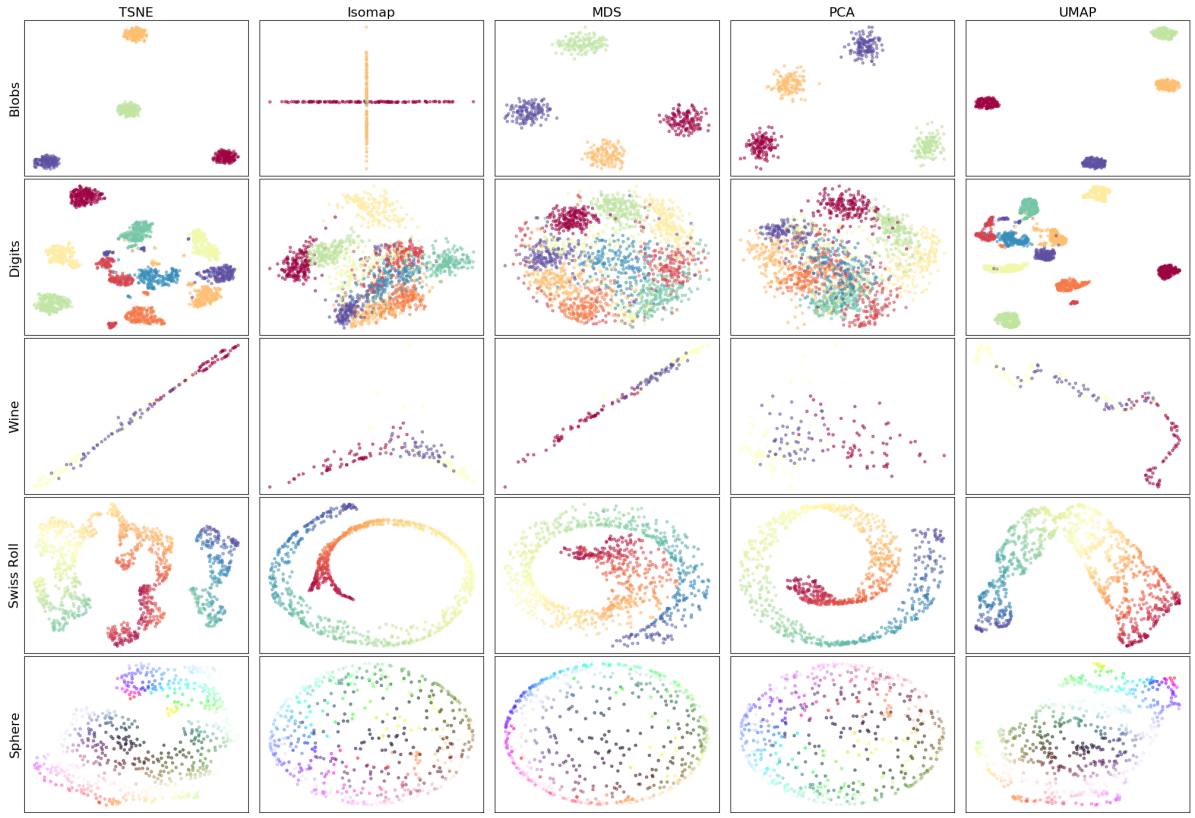
Now, we compute the 2D reductions for every reducer-dataset pair (this step is time-consuming):

```
reductions_and_labels = [(reducer(n_components=2,
      **args).fit_transform(data),
      labels)
    for data, labels in test_data
    for reducer, args in reducers
  ]
```

And we display the results:

```
n_rows = len(test_data)
n_cols = len(reducers)
fig = plt.figure(figsize=(16, 16))

fig.subplots_adjust(left=.02, right=.98, bottom=.001,
  top=.96, wspace=.05, hspace=.02)

ax_index = 1
ax_list = []

for reduction, labels in reductions_and_labels:
    ax = fig.add_subplot(n_rows, n_cols, ax_index)
    if isinstance(labels[0], tuple):
        # if labels are colours, use them
        ax.scatter(*reduction.T, s=10, c=labels, alpha=0.5)
    else:
        # otherwise, use "spectral" map from labels to colours
        ax.scatter(*reduction.T, s=10, c=labels,
            cmap="Spectral", alpha=0.5)
    ax_list.append(ax)
    ax_index += 1

plt.setp(ax_list, xticks=[], yticks=[])

for i in np.arange(n_rows) * n_cols:
    ax_list[i].set_ylabel(dataset_names[i // n_cols], size=16)

for i in range(n_cols):
    ax_list[i].set_xlabel(repr(reducers[i][0]()).split("(")[0],
        size=16)
    ax_list[i].xaxis.set_label_position("top")

fig.show()
```

Notice that Isomap removes exactly the wrong dimension in the swiss roll; $t-$SNE (and MDS to some extent) reduces the wine data down to one dimension (its true dimensionality) even though it was only asked for a reduction to two dimensions. UMAP manages to give the most sensible output for the swiss roll.



23.5 Exercises

Consider the datasets

- `GlobalCitiesPBI.csv`
- `2016collisionsfinal.csv`
- `polls_us_election_2016.csv`
- `HR_2016_Census_simple.xlsx`
- `UniversalBank.csv`.

and/or any other datasets of interest (as long as they have a sufficiently large number of predictors).

1. Establish 2-3 questions that you could try to answer with each dataset.
2. Based on the questions obtained in 1, provide 3-5 subsets of features that would do a good job of representing each dataset (use some of the methods described in this module, or other methods as needed).
3. Learn 3-5 reduced manifolds for each dataset (use some of the methods described in this module, or other methods as needed).
4. How would you validate your results?

Chapter References

[1] C.C. Aggarwal. *Data Mining: the Textbook*. Cham: Springer, 2015.

[2] C.C. Aggarwal, ed. *Data Classification: Algorithms and Applications*. CRC Press, 2015.

[3] C.C. Aggarwal and C.K. Reddy, eds. *Data Clustering: Algorithms and Applications*. CRC Press, 2014.

[4] David Duvenaud. 'Automatic Model Construction with Gaussian Processes'. PhD thesis. Computational and Biological Learning Laboratory, University of Cambridge, 2014.

[5] Y. Goldberg. *Neural Network Methods for Natural Language Processing*. Morgan and Claypool, 2017.

[6] Timothy Gowers, June Barrow-Green, and Imre Leader, eds. *The Princeton Companion to Mathematics*. Princeton University Press, 2008.

[7] F.E. Harrell. *Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis*. Springer Series in Statistics. Springer International Publishing, 2015.

[8] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, *2nd ed.* Springer, 2008.

[9] *Eta-squared*.

[10] 'Stopping stepwise: Why stepwise selection is bad and what you should use instead'. In: *towardsdatascience.com* (2018).

[11] *Interactive visualization to teach about the curse of dimensionality*.

[12] learntech. *Chi-squared test for nominal (categorical) data*. UWE Bristol.

[13] Y. LeCun, C. Cortes, and C.J.C. Burges. *The MNIST database of handwritten digits*.

[14] J. Leskovec, A. Rajamaran, and J.D. Ullman. *Mining of Massive Datasets*. Cambridge Press, 2014.

[15] Leland McInnes, John Healy, and James Melville. 'UMAP: Uniform manifold approximation and projection for dimension reduction'. In: *arXiv preprint* (2018).

[16] W. Morrissette. *Scotland, PA*. 2001.

[17] Natural Stat Trick. *Ottawa Senators @ Toronto Maple Leafs Game Log*. 2017.

[18] Sam T. Roweis and Lawrence K. Saul. 'Nonlinear Dimensionality Reduction by Locally Linear Embedding'. In: *Science* 290.5500 (2000), pp. 2323–2326.

[19] scikit-learn.org. *Manifold learning on handwritten digits*.

[20] scikit-learn.org. *Manifold Learning*.

[21] *Sens rally after blowing lead, beat Leafs to gain on Habs*. Associated Press. 2017.

[22] Alexander J. Smola and Risi Kondor. 'Kernels and Regularization on Graphs'. In: *Learning Theory and Kernel Machines*. Ed. by Bernhard Schölkopf and Manfred K. Warmuth. Springer Berlin Heidelberg, 2003, pp. 144–158.

[23] Yijun Sun and Dapeng Wu. 'A RELIEF Based Feature Extraction Algorithm'. In: Apr. 2008, pp. 188–195.

[24] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. 'A Global Geometric Framework for Nonlinear Dimensionality Reduction'. In: *Science* 290.5500 (2000), p. 2319.

[25] Tvtropes.org. *Laconic Macbeth*.

[26] Wikipedia. *Globalization and World Cities Research Network*.

[27] Wikipedia. *Macbeth*.

[28] Wikipedia. *Mutual information*.

[29] Wikipedia. *Pearson correlation coefficient*.

[30] Wikipedia. *Point biserial correlation coefficient*.

[31] Wikipedia. *t-distributed stochastic neighbor embedding*.

[32] Tong Zhang and Rie Kubota Ando. 'Analysis of Spectral Kernel Design based Semi-supervised Learning'. In: *Advances in Neural Information Processing Systems*. Ed. by Y. Weiss, B. Schölkopf, and J. Platt. Vol. 18. MIT Press, 2005.

[33] Z.A. Zhao and H. Liu. *Spectral Feature Selection for Data Mining*. CRC Press, 2011.