DATA UNDERSTANDING

DATA ANALYSIS

DATA SCIENCE

Volume 4: Techniques of Data Analysis

PATRICK BOILY

Data Understanding, Data Analysis, and Data Science (Course Notes)

Volume 4: Techniques of Data Analysis

Patrick Boily

May 2025

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 a 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 DUDADS. 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, May 2025
pboily@uottawa.ca

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:

Kevin Cheung *Mining Data Streams*

Youssouph Cissokho *Anomaly Detection and Outlier Analysis*

Soufiane Fadel *Anomaly Detection and Outlier Analysis*

Ehssan Ghashim *Bayesian Data Analysis*, *Queueing Systems*

Lani Haque *Text Analysis and Text Mining*

Andrew Macfie *Text Analysis and Text Mining*

Richard Millson *Anomaly Detection and Outlier Analysis*

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 reference manual at various levels in my teaching at the University of Ottawa (in the Department of Mathematics and Statistics).

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

- **MAT 4376/5314X** (*Special Topics in Probability and Statistics | Techniques of Data Analysis*) – Chapters 24–26 (and some material from ***The Practice of Data Visualization***);
- **MAT 4376/5314Y** (*Special Topics in Probability and Statistics | Applied Data Science*) – Chapters 27–28 (as well as Chapter 16 in DUDADS, Volume 2 and Chapter 29 in DUDADS, Volume 5).

I strongly encourage students to familiarize themselves with the contents of Chapters 1–11, 15, 19–23 (DUDADS, Volumes 1–3) before taking these courses, although it is not strictly necessary that they do so.

Contents

List of Figures

List of Tables

Queueing Models 24

by **Ehssan Ghashim** and **Patrick Boily**

Queueing theory focuses on waiting in lines (or queues). As a topic in operational research, it combines elements of a variety of quantitative disciplines, but it is rarely found in the data analyst's toolbox. In this chapter, we introduce the terminology and basic framework of queueing models (including Kendall-Lee notation, birth-death processes, and Little's formula), as well as the most commonly-used queueing system: $M/M/c$.

24.1 Background

Queueing theory is a branch of mathematics that studies and models the act of waiting in lines. The seminal paper on queueing theory [3] was published in 1909 by Danish mathematician A.K. Erlang; in it, he studied

> the problem of determining how many telephone circuits were necessary to provide phone service that would prevent customers from waiting too long for an available circuit. In developing a solution to this problem, he began to realize that the problem of minimizing waiting time was applicable to many fields, and began developing the theory further. Erlang's **switchboard problem** laid the path for modern queueing theory [1].

Queueing theory boils down to answering simple questions:

- How likely is it that objects/units/persons will queue up and wait in line?
- How long will the line be?
- How long will the wait be?
- How busy will the system be?
- How much capacity is needed to meet an expected level of demand?

Knowing how to think about these kinds of questions will help analysts and stakeholder anticipate **bottlenecks**. As a result, they will build systems and teams to be more efficient and more scalable, to have higher performance and lower costs, and to ultimately provide better service to their customers and end users.

Queueing theory also allows for the quantitative treatment of bottlenecks and their effect on performance. For instance, a question such as "how long will the wait be, on average?" will have an answer, but so will other

questions concerning the variability of wait times, the distribution of wait times, and the likelihood that a customer will receive extremely poor service, and so on [5].

Let us consider a simple example. Suppose a grocery store has a single checkout line and a single cashier. If, on average, one shopper arrives at the line to pay for their groceries every 5 minutes and if scanning, bagging, and paying takes 4.5 minutes, on average, would we expect customers to have to wait in line?

When the problem is presented this way, our intuition says that there should be no waiting in line, and that the cashier should be idle, on average, 30 seconds every 5 minutes, only being busy 90% of the time. No one ever has to wait before being served!

If you have ever been in a grocery store, however, you know that this is not what happens in reality; many shoppers will wait in line, and they will have to wait a long time before being processed.

Fundamentally, **queueing** happens for three reasons:

- **irregular arrivals** – shoppers do not arrive at the checkout line on a regular schedule; they are sometimes spaced far apart and sometimes close together, so they **overlap** (an overlap automatically causes queueing and waiting);
- **irregular job sizes** – shoppers do not all get processed in 4.5 minutes; someone shopping for a large family will require much more time than someone shopping only for themselves, for instance (when this happens, overlap is again a problem because new shoppers will arrive and be ready to check out while the existing ones are still in progress), and
- **waste** – lost time can never be regained; shoppers overlap because the second shopper arrived too soon, before the first had the time to finish being served, but looking at it the other way, perhaps it's not the second shopper's fault; perhaps the first shopper should have arrived earlier, but they wasted time reading a magazine while the cashier was idle! They missed their chance for quick service and, as a result, made the second shopper have to wait.

Irregular arrival times and job sizes are guaranteed to cause queueing. The only time there is no queueing is when the job sizes are uniform, the arrivals are timed evenly, and there is little enough work for the cashier to keep up with the arrival. Even when the cashier is barely busy, irregular arrivals or arrivals **in bursts** will cause some queueing.

In general, queueing gets worse when the following hold:

- **high utilisation** – the busier the cashier is, the longer it takes to recover from wasted time;
- **high variability** – the more variability in arrivals or job sizes, the more waste and the more overlap (queueing) occurs, and
- **insufficient number of servers** – fewer cashiers means less capacity to absorb arrival spikes, leading to more wasted time and higher utilisation.

In order to describe queues, we must first know and understand some useful probability distributions, as well as input and output processes.

24.2 Terminology

Queueing theory studies processes in terms of three key concepts:

- **customers** are the units of work that the system serves – a customer can be a real person, or it can be whatever the system is supposed to process and complete: a web request, a database query, a part to be milled by a machine, etc.;
- **servers** are the objects that do the processing work – a server might be the cashier at the grocery store, a web server, a database server, a milling machine, etc., and
- **queues** are where the units of work wait if the server is busy and can not start the work as they arrive – a queue may be a physical line, reside in memory, etc.

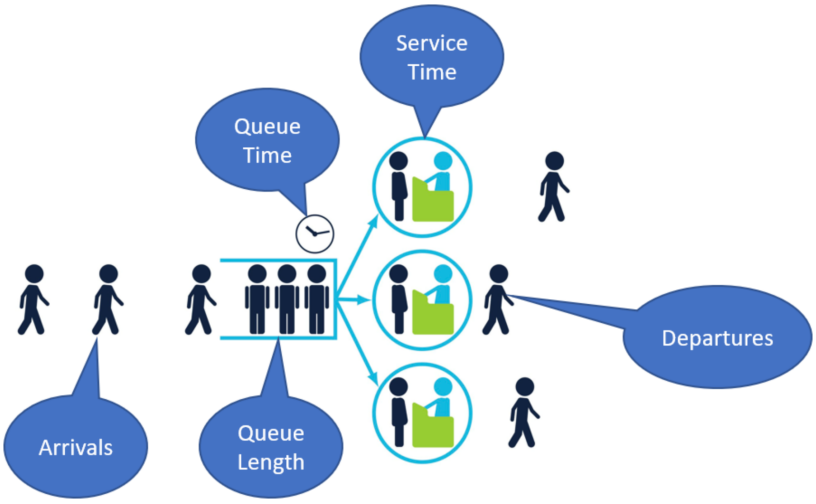


Figure 24.1: Components of a generic queueing system, by D. Hare.

Useful Distributions Three distributions play a central role in queueing theory: Poisson, exponential, and Erlang distributions.

Poisson Distribution The **Poisson distribution** counts the number of discrete events occurring in a fixed time period; it is closely connected to the exponential distribution, which describes the time between arrivals of the events. The Poisson distribution is a discrete distribution; the random variable can only take non-negative integer values. The exponential distribution is negative; the random variable can take any real (non-negative) value.

Consider the problem of determining the probability of n arrivals being observed during a time interval of length t, where the following assumptions are made:

- the probability that an arrival is observed during a small time interval (say of length ν) is proportional to the length of interval; let the proportionality constant be λ, so that the probability is $\lambda\nu$;
- the probability of two or more arrivals in a small interval is zero;

- the number of arrivals in any time interval is independent of the number of arrivals in other non-overlapping time intervals – for example, the number of arrivals occurring between times 5 and 25 does not provide information about the number of arrivals occurring between times 30 and 50.

Let $P(n;t)$ be the probability of observing n arrivals in a time interval of length t. Then, for some $\lambda > 0$,

$$P_\lambda(n;t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}, \; n = 0, 1, 2, \ldots$$

is the probability mass function of the Poisson distribution for the discrete random variable n – the number of arrivals – for a given length of time interval t (see Figure 24.2).

Example: on average, 50 customers arrive in a coffee shop every hour. What is the probability that exactly 20 customers will arrive in a 30-minute period, if the arrivals follow a Poisson distribution?

Solution: given $\lambda = 50$ customers per hour, $t = 30$ min $= 0.5$ hr and $n = 20$, we have

$$P_{50}(20; 0.5) = \frac{(50 \cdot 0.5)^{20}}{20!} e^{-50 \cdot 0.5} \approx 5.2\%.$$

We can evaluate the probability directly in `R` *via*

```
n=20
lambda=50
t=0.5
dpois(n,lambda*t)
```

```
[1] 0.05191747
```

In a queueing system, such arrivals are referred to as **Poisson arrivals**. The time between successive arrivals is called the **inter-arrival time**.

Exponential Distribution If the number of arrivals in a given time interval follows a Poisson distribution with parameter λt, the inter-arrival times follow an **exponential distribution** with probability density function

$$f_\lambda(t) = \lambda e^{-\lambda t}, \text{ for } t > 0,$$

and the probability $P(W \leq t)$ that a customer's waiting time W is smaller than the length of the time interval t is

$$P(W \leq t) = 1 - e^{-\lambda t}$$

(see Figure 24.2). We would write $W \sim \text{Exp}(\lambda)$.

Example: a fast-food restaurant's manager's manager observes that an average of 9 customers are served by a waiter in a one-hour time period. Assuming that the service time follows an exponential distribution, what is the probability that a customer will be served within 15 minutes?

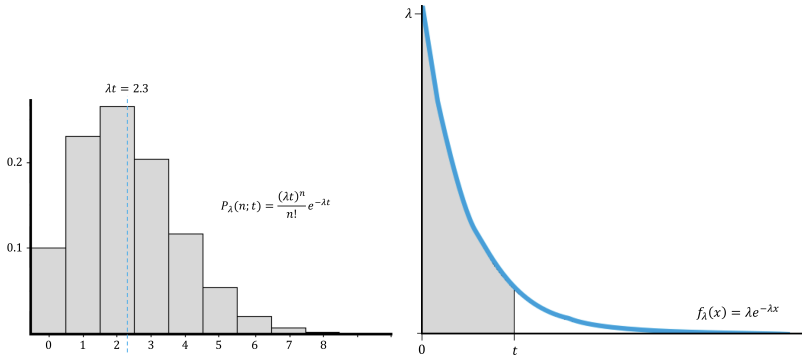


Figure 24.2: Poisson (with $\lambda t = 2.3$) and exponential distributions (with parameter λ). The shaded area (right) represents the probability that a customer will wait up t time units.

Solution: let w be the average waiting time. Given $\mu = 9$ customers per hours, $t = 15$ min $= 0.25$ hr, we have

$$P(w \leq 15 \text{ min}) = 1 - e^{-9 \cdot 0.25} \approx 89.5\%.$$

We can evaluate the probability directly in R.

```
t=0.25
mu=9
pexp(t,rate=mu)
```

```
[1] 0.8946008
```

In general, if the arrival rate is **stationary**, if **bulk** arrivals (two or more simultaneous arrivals) cannot occur, and if past arrivals do not affect future arrivals, then inter-arrival times follow an exponential distribution with parameter λ, and the number of arrivals in any interval of length t is Poisson with parameter λt.

One of the most attractive features of the exponential distribution relating to inter-arrival times is that it is **memoryless** – if X follows an exponential distribution, then for all non-negative values of t, h,

$$P(X \geq t + h \mid X \geq t) = P(X \geq h).$$

No other density function satisfies this property [8].

The memoryless property of the exponential distribution is important because it implies that the probability distribution of the time until the next arrival is independent of the time since the last arrival. This is clearly not always the case – imagine if that was so when waiting for public transportation!

For instance, if we know that at least t time units have elapsed since the last arrival, then the distribution of the time h until the next arrival is independent of t. If $h = 4$, say, then we must have

$$P(X > 9 \mid X > 5) = P(X > 7 \mid X > 3) = P(X > 4).$$

Example: the time W a customer spends waiting in a bank queue is exponentially distributed with mean $\lambda = 10$ min, say. If they've already

waited 10 minutes, what is the probability that they will have had to wait more than 15 minutes in total, when all is said and done?

Solution: thanks to the memory-less property of the exponential distribution, we have

$$P(W > 15 \mid W > 10) = P(W > 15{-}10 = 5) = \exp(-5/\lambda) = \exp(-1/2) \approx 60.6\%.$$

We can evaluate the probability directly in R.

```
w=5
lambda=10
1-pexp(w,rate=1/lambda)
```

```
[1] 0.6065307
```

Erlang Distribution The exponential distribution is not always an appropriate model of inter-arrival times, however (perhaps the process should not be memoryless, say).

A common alternative is to use the **Erlang** distribution $\mathscr{E}(R, k)$, a continuous random variable with **rate** and **shape** parameters $R > 0$ and $k \in \mathbb{Z}^+$, respectively, whose probability density function is

$$f_{R,k}(t) = \frac{R(Rt)^{k-1}e^{-Rt}}{(k-1)!},\ t \geq 0.$$

If $k = 1$, the Erlang distribution reduces to an exponential distribution with parameter R. It can further be shown that if $X \sim \mathscr{E}(R, k)$, where $R = k\lambda$, then $X \sim X_1 + X_2 + \cdots + X_k$, where each $X_i \sim \text{Exp}(R)$ is an independent random variable.

When we model the inter-arrival process as an Erlang distribution $\mathscr{E}(k\lambda, k)$, we are really saying that it is equivalent to customers going through k **phases** (each of which is memoryless) before being served.

For this reason, the shape parameter is often referred to as the number of phases of the Erlang distribution [7].

24.2.1 Input/Arrival Processes

The input process is usually called the **arrival process**. Arrivals are called **customers**. In the models under consideration, we assume that arrivals cannot be simultaneous (this might be unrealistic when modeling arrivals at a restaurant, say). If simultaneous arrivals are possible (in theory and/or in practice), we say that **bulk arrivals are allowed**.

Usually, we assume that the arrival process is **unaffected by the number of customers** in the system. In the context of a bank, this would imply that whether there are 500 or 5 people at the bank, the process governing arrivals remains unchanged. There are two common situations in which the arrival process may depend on the number of customers present. The first occurs when arrivals are drawn from a small population – the

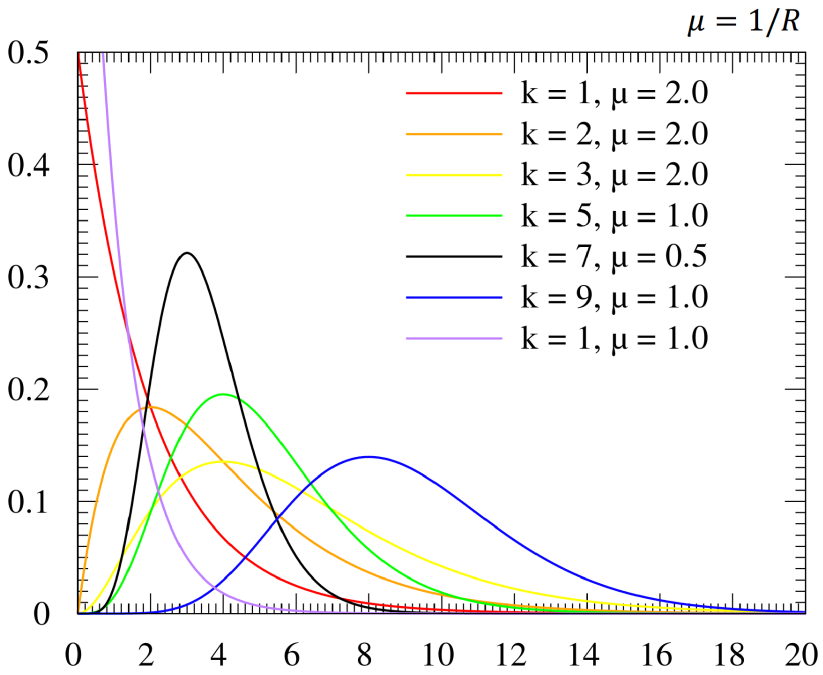


Figure 24.3: Probability distribution functions for various Erlang random variables [Wikipedia].

so-called **finite source models** – if all members of the population are already in the system, there cannot be another arrival!

Another such situation arises when the rate at which customers arrive at the facility decreases when the facility becomes too crowded. For example, when customers see that a restaurant's parking lot is full, they might very well decide to go to another restaurant or forego eating out altogether. If a customer arrives but fails to enter the system, we say that the customer has **balked**.

24.2.2 Output/Service Processes

To describe the output process (often called the **service process**) of a queueing system, we usually specify a probability distribution – the **service time distribution** – which governs the customers' service time.

In most cases, we assume that the service time distribution is independent of the number of customers present in the system. This implies, for example, that the server does not work faster when more customers are present. We can distinguish two types of servers: **in parallel** and **in series**.

Servers are **in parallel** if they all provide the same type of service and a customer only needs to pass through one of them to complete their service. For example, the tellers in a bank are usually arranged in parallel; typically, customers only need to be serviced by one teller, and any teller can perform the desired service.

Servers are **in series** if a customer must pass through several servers before their service is complete. An assembly line is an example of such a queueing system. Input and output processes occur in a variety of situations:

- **situation:** purchasing Blue Jays tickets at the Rogers Centre
  input: baseball fans arrive at the ticket office
  output: tellers serve the baseball fans
- **situation:** pizza parlour
  input: requests for pizza delivery are received
  output: pizza parlour prepares and bakes pizzas, and sends them to be delivered
- **situation:** government service centre
  input: citizen/residents enter the service centre
  output: receptionist assigns them to a specific queue based on their needs:
  - *input:* citizen/residents enter a specific queue based on their needs
    output: public servant addresses their needs
- **situation:** hospital blood bank
  input: pints of blood arrive
  output: patients use up pints of blood
- **situation:** garage
  input: cars break down and are sent to the garage for repairs
  output: cars are repaired by mechanics and sent back on the streets

The relevant computations are fairly easy to execute, as the following examples demonstrate.

Example: On average, 4.6 customers enter a coffee shop each hour. If the arrivals follow a Poisson process, what is the probability that at most two customers will enter in a 30 minute period?

Solution: since 30 min = 0.5 hr, we have

$$\begin{aligned} P_{\lambda=4.6}(n \le 2; t = 0.5) &= P_{4.6}(0, 0.5) + P_{4.6}(1, 0.5) + P_{4.6}(2, 0.5) \\ &= e^{-4.6 \cdot 0.5} \left[\frac{(4.6 \cdot 0.5)^0}{0!} + \frac{(4.6 \cdot 0.5)^1}{1!} + \frac{(4.6 \cdot 0.5)^2}{2!} \right] \\ &\approx 0.5960; \end{aligned}$$

the corresponding Poisson distribution is shown in Figure 24.2.

We can evaluate the probability directly in R.

```
n=2
lambda=4.6
t=0.5
ppois(n,lambda*t)
```

```
[1] 0.5960388
```

Example: in a fast food restaurant, a cashier serves on average 9 customers in a one-hour time period. If the service time follows an exponential distribution, what percentage of customers will be served in 10 minutes or less? After 30 minutes?

Solution: since 1 hr $=$ 60 mins, we have $\mu = 9$ customers/60 minutes, and so

$$P(W \leq 10/60) = 1 - e^{-9 \cdot 10/60} \approx 0.7769$$
$$P(W > 30/60) = e^{-9 \cdot 30/60} \approx 0.0111.$$

24.2.3 Queue Discipline

To describe a queueing system completely, we must also describe the **queue discipline** and the manner in which customers **join lines**. The queue discipline describes the method used to determine the order in which customers are served:

- the most common queue discipline is the **first come, first served** (FCFS) discipline, in which customers are served in the order of their arrival, as one would expect to see in an Ottawa coffee shop;
- under the **last come, first served** (LCFS) discipline, the most recent arrivals are the first to enter service; for example, if we consider exiting from an elevator to be the service, then a crowded elevator illustrates such a discipline;
- sometimes the order in which customers arrive has no effect on the order in which they are served; this would be the case if the next customer to enter service is randomly chosen from those customers waiting for service, a situation referred to as **service in random order** (SIRO) discipline; when callers to an inter-city bus company are put on hold, the luck of the draw often determines which caller will next be serviced by an operator;
- finally, **priority** discipline classifies each arrival into one of several categories, each of which is assigned a priority level (a **triage** process); within each priority level, customers enter the queue on a FCFS basis; such a discipline is often used in emergency rooms to determine the order in which customers receive treatment, and in copying and computer time-sharing facilities, where priority is usually given to jobs with shorter processing times.

24.2.4 Method Used by Arrivals to Join Queue

Another important factor for the behaviour of the queueing system is the **method** used by customers to determine which line to join. For example, in some banks, customers must join a single line, but in other banks, customers may choose the line they want to join.

When there are several lines, customers often join the shortest line. Unfortunately, in many situations (such as at the supermarket), it is difficult to define the shortest line. If there are several lines at a queueing facility, it is important to know whether or not customers are allowed to **switch**, or jockey, between lines. In most queueing systems with multiple lines, jockeying is permitted, but jockeying at a custom inspection booth would not be recommended (if it is even allowed), for instance.

24.3 Queueing Theory Framework

There is a standard notation that is used to describe large families of queueing systems: the **Kendall-Lee notation** [4].

24.3.1 Kendall-Lee Notation

queueing systems can be described *via* six characteristics:

$$x_1/x_2/x_3/x_4/x_5/x_6.$$

The 1st characteristic x_1 specifies the nature of the **arrival process**. The following standard abbreviations are used:

M	inter-arrival times are independent identically distributed (iid) exponentials
D	inter-arrival times are iid and deterministic
E_k	inter-arrival times are iid Erlangs with shape parameter k
G	inter-arrival times are iid and governed by some general distribution

The 2nd characteristic x_2 specifies the nature of the **service times**:

M	service times are iid and exponential
D	service times are iid and deterministic
E_k	service times are iid Erlang with shape parameter k
G	service times are iid and follow some general distribution

The 3rd characteristic x_3 represents the **number of parallel servers**.

The 4th characteristic x_4 describes the **queue discipline**:

FCFS	first come, first served
LCFS	last come, first served
SIRO	service in random order
GD	general queue discipline

The 5th characteristic x_5 specifies the **maximum allowable number of customers in the system**.[1]

1: Including customers who are waiting and customers who are in service.

The 6th characteristic x_6 gives the **size of the population** from which customers are drawn. Unless the number of potential customers is of the same order of magnitude as the number of servers, the population size is considered to be infinite.

In many important models $x_4/x_5/x_6$ is GD/∞/∞.[2] As an example, $M/M/3$/FCFS/$20/\infty$ could represent a bank with 3 tellers, exponential arrival times, exponential service times, a "first come, first served" queue discipline, a total capacity of 20 customers, and an infinite population pool from which to draw. The situation is illustrated in Figure 24.4.

2: When that is the case, the string is often omitted.

Examples: here are some commonly-used/studied queueing systems:

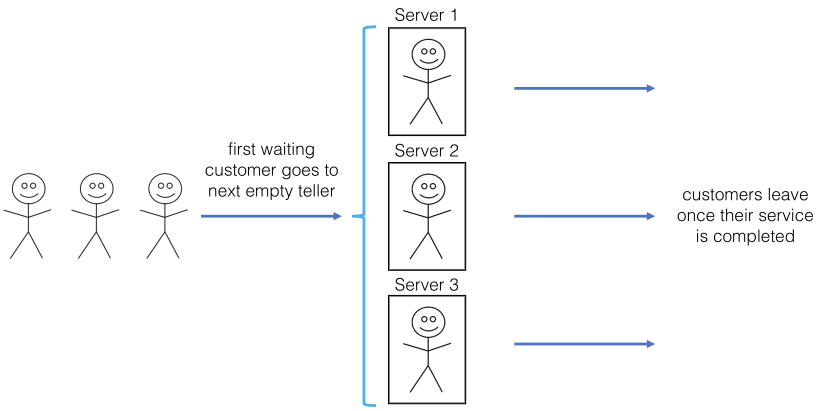


Figure 24.4: Single line at bank with three tellers – $M/M/3/\text{FCFS}/20/\infty$.

Name	Notation	Example
simple system	$M/M/1$	customer service desk in a small store
multi-server system	$M/M/c$	airline ticket counter
constant service	$M/D/1$	automated car wash
general service	$M/G/1$	auto repair shop
limited capacity	$M/M/1/N$	barber shop with N waiting seats

24.3.2 Birth-Death Processes

The **state** of a queueing system at time t is defined to be the number of customers in the queueing system, either waiting in line or in service, at time t. At $t = 0$, the state of the system is the initial number of customers in the system. This state is worth recording because it clearly affects the state at future times t.

Knowing this, we define $P_{i,j}(t)$ as the probability that the state at time t is j, given that the state at $t = 0$ was i. For large t, $P_{i,j}(t)$ becomes independent of i and approaches a limit π_j. This limit is known as the **steady-state** of state j.

It is generally quite difficult to determine the steps of arrivals and services that lead to a steady-state π_j. Likewise, starting from an early t, it is difficult to determine exactly when a system will reach its steady state π_j, if such a state even exists.

For simplicity's sake, when a queueing system is studied, we begin by assuming that the steady-state has already been reached. A **birth-death process** is a Markov process in which states are indexed by non-negative integers, and transitions are only permitted between "neighbouring" states. After a "birth", the state increases from n to $n + 1$; after a "death", the state decreases from m to $m - 1$.

Typically, we denote the set of birth rates and death rates by λ_n and μ_m, respectively (see Figure 24.5).

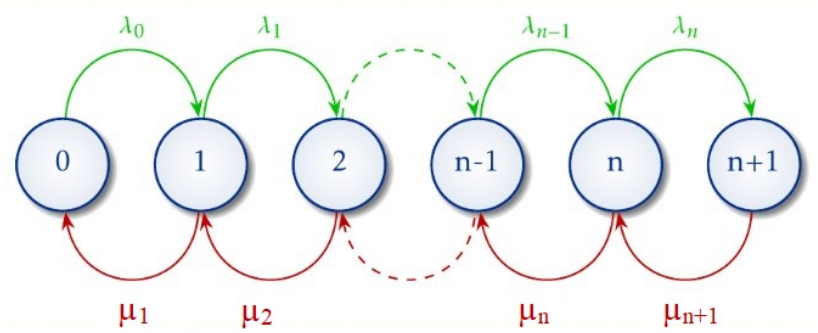


Figure 24.5: Birth-death process; queueing states indexed by integers; birth rates and death rates indicated by λ_n and μ_m, respectively (source unknown).

Pure birth processes are those for which $\mu_m = 0$ for all m; **pure death** processes, those for which $\lambda_n = 0$ for all n. The **steady-state solution** of a birth-death process, i.e., the probability π_n of being in state n, *can* actually be computed:

$$\pi_n = \pi_0 \frac{\lambda_0 \lambda_1 \cdots \lambda_{n-1}}{\mu_1 \mu_2 \cdots \mu_n}, \quad \text{for } n = 1, 2, \ldots$$

where π_0 is the probability of being in state 0 (i.e., without users). It can further be shown [5] that:

$$\pi_0 = \frac{1}{1 + \sum_{n=1}^{\infty} \prod_{j=0}^{n-1} \frac{\lambda_j}{\mu_{j+1}}}.$$

24.3.3 Little's Queueing Formula

It is often the case that clients and end users are interested in determining the **amount of time** that a typical customer spends in the queueing system. Let W be the **expected waiting time** spent in the queueing system, including time in line plus time in service, and W_q be the **expected time a customer spends waiting in line**.

Both W and W_q are computed under the assumption that the steady state has been reached. By using a powerful result known as **Little's queueing formula**, W and W_q are easily related to the number of customers in the queue and those waiting in line. For any queueing system (or any subset of a queueing system), consider the following quantities:

- λ = average number of arrivals entering the system per unit time;
- L = average number of customers present in the queueing system;
- L_q = average number of customers waiting in line;
- L_s = average number of customers in service;
- W = average time a customer spends in the system;
- W_q = average time a customer spends in line, and
- W_s = average time a customer spends in service.

Customers in the system can only be found in the queue or being serviced, so that $L = L_q + L_s$ and $W = W_q + W_s$. In these definitions, all averages are **steady-state averages**.

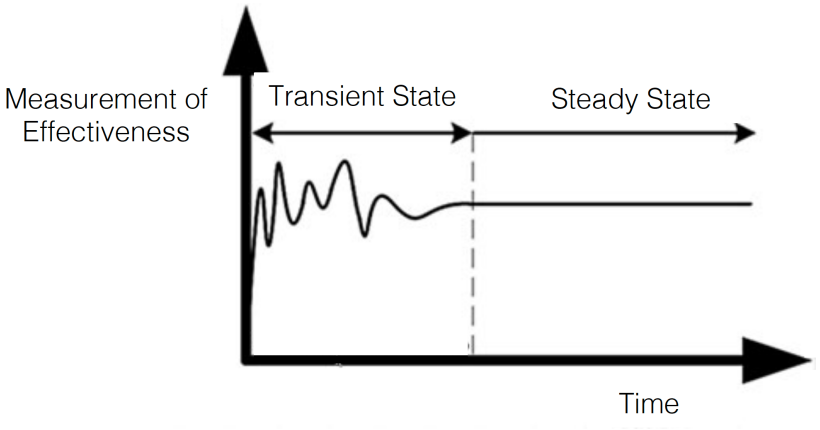


Figure 24.6: Schematics of steady state vs. transient behaviour (source unknown).

For most queueing systems in which a steady-state exists, **Little's queueing formula** are summarized by:

$$L = \lambda W, \quad L_q = \lambda W_q, \quad \text{and} \quad L_s = \lambda W_s.$$

Example: if, on average, 46 customers enter a restaurant each hour it is opened, and if they spend, on average, 10 minutes (1/6 hours) waiting to be served, then we should expect $46 \cdot 1/6 \approx 7.7$ customers in the queue at all time (on average).

24.4 $M/M/1$ Queueing Systems

We now discuss the **simplest non-trivial** queueing system.

24.4.1 Basics

An $M/M/1/\text{GD}/\infty/\infty$ queueing system has exponential inter-arrival times, exponential service times, and a single server. It can be modeled as a birth-death process with

$$\begin{aligned} \lambda_j &= \lambda, \ j = 0, 1, 2, \dots \\ \mu_0 &= 0 \\ \mu_j &= \mu, \ j = 1, 2, 3, \dots \end{aligned}$$

Substituting these rates in the steady-state solution of a birth-death process yields

$$\pi_j = \frac{\lambda^j \pi_0}{\mu^j} = \rho^j \pi_0,$$

where $\rho = \lambda/\mu$ is the **traffic intensity** of the system.

Since the system has to be in exactly one of the states at any given moment, the sum of all probabilities is 1:

$$\pi_0 + \pi_1 + \pi_2 + \cdots = \pi_0(1 + \rho + \rho^2 + \cdots) = 1.$$

If $0 \leq \rho < 1$, the infinite series converges to $\frac{1}{1-\rho}$ from which we derive

$$\pi_0 \cdot \frac{1}{1-\rho} = 1 \implies \pi_0 = 1 - \rho \implies \pi_j = \rho^j \pi_0 = \rho^j (1 - \rho)$$

as the **steady-state probability of state** j.

If $\rho \geq 1$, the infinite series diverges and no steady-state exists. Intuitively, this happens when $\lambda \geq \mu$, that is, if the arrival rate is greater than the service rate, then the state of the system grows without bounds and the queue is never cleared. From this point on, we assume $\rho < 1$ to guarantee that the steady-state probabilities π_j exist, from which we can determine several quantities of interest.

Assuming that the steady state has been reached, it can be shown that L, L_s, and L_q are given respectively by:

$$L = \frac{\lambda}{\mu - \lambda} = \frac{\rho}{1 - \rho}, \quad L_s = \rho, \quad L_q = \frac{\rho^2}{1 - \rho}.$$

Using Little's queueing formula, we can also solve for W, W_s, and W_q by dividing each of the corresponding L values by λ:

$$W = \frac{1}{\mu - \lambda}, \quad W_s = \frac{1}{\mu}, \quad W_q = \frac{\lambda}{\mu(\mu - \lambda)}.$$

Note that, as expected, both $W, W_q \to +\infty$ when $\rho \to 1$. On the other hand, $W_q \to 0$ and $W \to \frac{1}{\mu}$ (the **mean service time**) as $\rho \to 0$.

Example: (based on [9]) an average of 10 cars arrive at a single-server drive-in teller every hour. Assume that the average customer is served in 4 minutes, and that both inter-arrival times and service times are exponentially distributed.

1. What is the probability that the teller is idle?
2. Excluding the car that is being served, what is the average number of cars waiting in line at the teller?
3. What is the average amount of time a drive-in customer spends in the bank parking lot (including time in service)?
4. On average, how many customers per hour are served by the teller?

Solution: by assumption, we are dealing with an $M/M/1/\text{GD}/\infty/\infty$ queueing system for which $\lambda = 10$ cars/hr and $\mu = 15$ cars/hr, and as such $\rho = 10/15 = 2/3$.

1. The teller is idle one third of the time on average because $\pi_0 = 1 - \rho = 1/3$.
2. There are $L_q = \rho^2/(1 - \rho) = 4/3$ cars waiting in line for the teller.
3. We know that $L = \lambda/(\mu - \lambda) = 10/(15 - 10) = 2$, and so $W = L/\lambda = 0.2$ hr $= 12$ min.
4. If the teller were always busy, it would serve an average of $\mu = 15$ customers per hour. From part 1., we know that the teller is only busy two-thirds of the time, thus during each hour, the teller serves an average of $15 \cdot 2/3 = 10$ customers. This is reasonable since, in a steady-state, 10 customers are arriving each hour and 10 customers must leave the system every hour.

Example: (based on [6]) suppose that all car owners fill up when their tanks are exactly half full. On average, 7.5 customers arrive every hour at a single-pump gas station. It takes an average of 4 minutes to fuel a car.

Assume that inter-arrival times and service times are both exponential.

1. What are the values of L and W in this scenario?
2. Suppose that a gas shortage occurs and panic buying takes place. To model this phenomenon, assume that all car owners now purchase gas when their tanks are exactly three-quarters full. Since each car owner is now putting less gas into the tank during each visit to the station, we assume that the average service time has been reduced to 10/3 minutes. How has panic buying affected the values of L and W?

Solution: by assumption, we again have an $M/M/1/\text{GD}/\infty/\infty$ queueing system, with $\lambda = 7.5$ cars/hr and $\mu = 60/4 = 15$ cars/hr. Thus, $\rho = 7.5/15 = 1/2$.

1. By definition, $L = \lambda/(\mu - \lambda) = 7.5/(15 - 7.5) = 1$ and $W = 1/7.5 \approx 0.13$ hr $= 7.8$ min. Hence, in this situation, everything is under control, and long lines appear to be unlikely.
2. Under the panic buying scenario, $\lambda = 2(7.5) = 15$ cars/hr as each car owner now fills up twice as often, and $\mu = 60 \cdot 3/10 = 18$ cars/hr, so $\rho = \lambda/\mu = 5/6$. In that scenario,

$$L = \frac{\rho}{1-\rho} = 5 \text{ cars}, \quad \text{and} \quad W = \frac{L}{\lambda} = \frac{5}{15} = 20 \text{ min}.$$

Thus, panic buying has more than doubled the wait time in line. In a $M/M/1$ queueing system, we have

$$L = \frac{\rho}{1-\rho} = -1 + \frac{1}{1-\rho},$$

and it is easy to see that $L \to \infty$ as $\rho \to 1$. The 5−fold increase in L when ρ jumps from $1/2$ to $5/6$ (with accompanying jumps in W) illustrates that fact.

ρ	L in a $M/M/1$ queue
0.30	0.43
0.60	1.50
0.80	4.00
0.90	9.00
0.95	19.00
0.99	99.00

24.4.2 Limited Capacity

In the real world, queues never become infinite – they are limited due to requirements of space and/or time, or service operating policy. Such a queueing model falls under the purview of **finite queues**.

Finite queue models restrict the number of customers allowed in the service system. Let N represent the maximum allowable number of customers in the system. If the system is at **capacity**, the arrival of a $(N+1)^{\text{th}}$ customer results in a failure to enter the queue – the customer is assumed to balk and depart without seeking service.

Finite queues can also be modeled as a birth-death process, but with a slight modification in its parameters:

$$\begin{aligned} \lambda_j &= \lambda, \ j = 0, 1, 2, \ldots, N-1 \\ \lambda_N &= 0, \ \mu_0 = 0 \\ \mu_j &= \mu, \ j = 1, 2, 3, \ldots, N. \end{aligned}$$

The restriction $\lambda_N = 0$ is what sets this model apart from the $M/M/1/\infty$. It makes it impossible to reach a state greater than N. Because of this

restriction, a steady-state always exist because even if $\lambda \geq \mu$, there can never be more than N customers in the system.

Mathematically, this has the effect of replacing the infinite series linking the π_j's by a finite geometric series, which always converges:

$$\pi_0 + \pi_1 + \cdots + \pi_N = \pi_0(1 + \rho + \cdots + \rho^N) = 1,$$

from which we can derive

$$\pi_0 \cdot \frac{1-\rho^{N+1}}{1-\rho} = 1 \implies \pi_0 = \frac{1-\rho}{1-\rho^{N+1}}$$
$$\implies \pi_j = \begin{cases} \rho^j \frac{1-\rho}{1-\rho^{N+1}} & \text{for } j = 0, \ldots, N \\ 0 & \text{for } j > N \end{cases}$$

Since $L = \sum_{j=0}^{N} j \cdot \pi_j$ (why?),

$$L = \frac{\rho[1 + N\rho^{N+1} - (N+1)\rho^N]}{(1-\rho)\left(1-\rho^{N+1}\right)}$$

when $\lambda \neq \mu$. As in $M/M/1/\infty$, $L_s = \lambda(1-\pi_0)$, and $L_q = L - L_s$.

In a finite capacity model, only $\lambda - \lambda\pi_N = \lambda\,(1-\pi_N)$ arrivals per unit time actually enter the system on average (λ arrive, but $\lambda\pi_N$ find the system full). With this fact,

$$W = \frac{L}{\lambda\,(1-\pi_N)} \quad \text{and} \quad W_q = \frac{L_q}{\lambda\,(1-\pi_N)}.$$

What does that look like in practice?

Example: consider a one-man barber shop with a total of 10 seats. Assume, as has always been the case so far (but need not be), that inter-arrival times are exponentially distributed with an average of 20 prospective customers arriving each hour at the shop. Those customers who find the shop full do not enter (perhaps they do not like standing). The barber takes an average of 12 minutes to cut each customer's hair; assume that haircut times are also exponentially distributed.

1. On average, how many haircuts per hour will the barber complete?
2. On average, how much time will be spent in the shop by a customer who enters?

Solution:

1. A fraction π_{10} of all arrivals will find the shop full, so that only an average of $\lambda\,(1-\pi_{10})$ will actually enter the shop each hour. All entering customers receive a haircut, so the barber will give an average of $\lambda\,(1-\pi_{10})$ haircuts per hour. In this scenario, $N = 10$, $\lambda = 20$ customers/hr, and $\mu = 60/12 = 5$ customers/hr. Thus $\rho = 20/5 = 4$ and we have

$$\pi_0 = \frac{1-\rho}{1-\rho^{N+1}} = \frac{1-4}{1-4^{11}} \approx 7.15 \times 10^{-7} \text{ and}$$
$$\pi_{10} = 4^{10}\pi_0 = \frac{3}{4} \text{ (verify!).}$$

In that case, an average of $20(1 - 3/4) = 5$ customers per hour will receive haircuts. This means that an average of $20 - 5 = 15$ prospective customers per hour will not enter the shop.

2. To determine W, we must first compute

$$L = \frac{4[1 + (10)4^{11} - (11)4^{10}]}{(1-4)(1-4^{11})} = 9.67.$$

Using the formulas described above, we obtain

$$W = \frac{L}{\lambda(1 - \pi_{10})} = \frac{9.67}{5} = 1.93 \text{ hr}.$$

This barber shop is quite crowded – the barber would be well-advised to hire at least one more worker!

But what *would* be the effect of hiring a second barber?

In order to answer this question, we need to look into $M/M/c$ queueing systems.

24.5 $M/M/c$ Queueing Systems

An $M/M/c/\text{GD}/\infty$ queueing system also has exponential inter-arrival and service times, with rates λ and μ, respectively. What sets this system apart is that there are now $c > 1$ servers willing to serve from a single line of customers, perhaps like one would find in a bank (see Figure 24.7).

If $j \leq c$ customers are present in the system, then every customer is being served and there is no wait time; if $j > c$ customers are in the system, then c customers are being served and the remaining $j - c$ customers are waiting in the queue. To model this as a birth-death process, we have to observe that the death rate is dependent on how many servers are actually being used.

If each server completes service at a rate of μ (which may not be the case in practice as there might be variations in servers, at least for human servers), then the **actual death rate** is $\mu\times$ the number of customers actually being served. The parameters for this process are

$$\lambda_n = \lambda, \ n = 0, 1, 2, \ldots$$
$$\mu_n = \begin{cases} n\mu, & n = 0, 1, 2, \ldots, c \\ c\mu, & n = c+1, c+2, \ldots \end{cases}$$

The traffic intensity for the $M/M/c$ system is $\rho = \lambda/(c\mu)$ and the steady-state solution is

$$\pi_n = \begin{cases} \frac{(c\rho)^n}{n!}\pi_0, & 1 \leq n \leq c \\ \frac{c^c\rho^n}{c!}\pi_0, & n \geq c \end{cases}$$

where

$$\pi_0 = \left[1 + \frac{(c\rho)^c}{c!(1-\rho)} + \sum_{n=1}^{c-1}\frac{c\rho^n}{n!}\right]^{-1}.$$

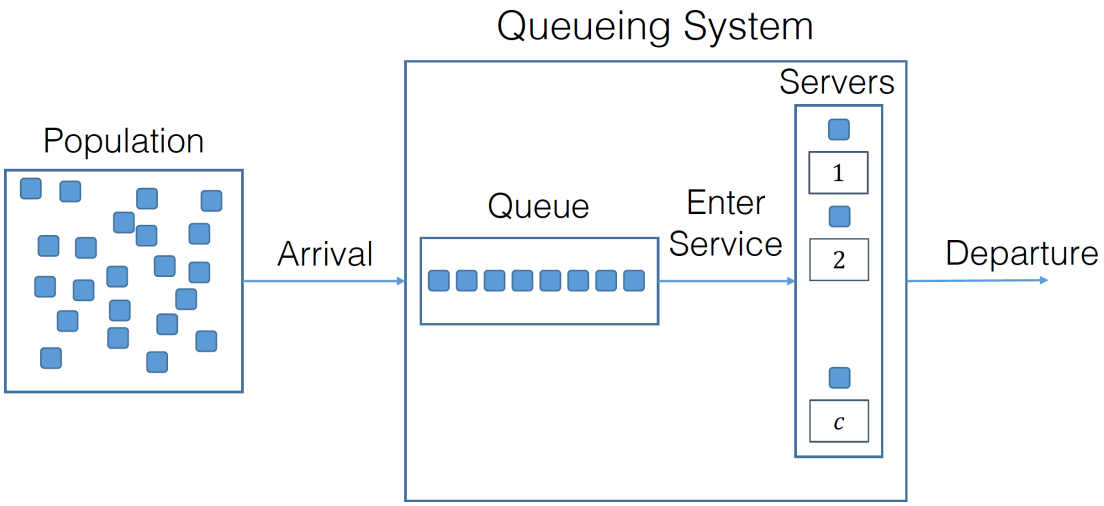


Figure 24.7: Generic $M/M/c$ queue.

Note that, as was the case in a $M/M/1$ system, if $\rho \geq 1$, there can be no steady state – in other words, if the arrival rate is at least as large as the maximum possible service rate ($\lambda \geq c\mu$), then the system "blows up".

There might be a desire to ensure that customers do not wait in line an inordinate amount of time, but there might also be a desire to minimize the amount of time for which at least one of the server is idle. In a $M/M/c$ queueing system, this steady-state probability is given by

$$P(n \geq c) = \frac{(c\rho)^c}{c!\,(1-\rho)}\pi_0.$$

This table shows the probabilities $P(n \geq c)$ that all servers are busy in an $M/M/c$ system for $c = 2, \ldots, 7$ and $0.1 \leq \rho \leq 0.95$ [9, p.1088].

ρ	$c=2$	$c=3$	$c=4$	$c=5$	$c=6$	$c=7$
.10	.02	.00	.00	.00	.00	.00
.20	.07	.02	.00	.00	.00	.00
.30	.14	.07	.04	.02	.01	.00
.40	.23	.14	.09	.06	.04	.03
.50	.33	.24	.17	.13	.10	.08
.55	.39	.29	.23	.18	.14	.11
.60	.45	.35	.29	.24	.20	.17
.65	.51	.42	.35	.30	.26	.21
.70	.57	.51	.43	.38	.34	.30
.75	.64	.57	.51	.46	.42	.39
.80	.71	.65	.60	.55	.52	.49
.85	.78	.73	.69	.65	.62	.60
.90	.85	.83	.79	.76	.74	.72
.95	.92	.91	.89	.88	.87	.85

Cumbersome calculations, using $W_s = \frac{1}{\mu}$, yield

$$L_q = \frac{\rho}{1-\rho}P(n \geq c), \quad W_q = \frac{L_q}{\lambda}, \quad W = \frac{1}{\mu} + W_q, \quad L = \frac{\lambda}{\mu} + L_q.$$

Example: consider, for instance, a bank with two tellers. An average of 80 customers arrive at the bank each hour and wait in a single line for an idle teller. For this specific bank, the average service time is 1.2 minutes. Assume that inter-arrival times and service times are exponential. Determine:

1. The expected number of customers in the bank.
2. The expected length of time a customer spends in the bank.
3. The fraction of time that a particular teller is idle.

Solution: we are dealing with an $M/M/2$ system with $\lambda = 80$ customers/hr and $\mu = 50$ customers/hr. Thus, $\rho = \frac{80}{2 \cdot 50} = 0.80 < 1$ and the steady-state exists.

1. From the above table, $P(n \geq 2) = 0.71$, from which we compute

$$L_q = P(n \geq 2) \cdot \frac{0.8}{1 - 0.8} = 2.84 \text{ customers}$$
$$L = \frac{80}{50} + L_q = 4.44 \text{ customers.}$$

2. We know that $W = \frac{L}{\lambda} = \frac{4.44}{80} = 0.055 \text{ hr } = 3.3 \text{ min.}$
3. To determine the fraction of time that a particular server is idle, note that tellers are idle during all moments when $n = 0$, and half the time (by symmetry) when $n = 1$. The probability that a server is idle is thus given by $\pi_0 + 0.5\pi_1$. But

$$\pi_0 = \left[1 + \frac{(2 \cdot 0.8)^2}{2!\,(1 - .8)} + \sum_{n=1}^{2-1} \frac{2 \cdot 0.8^n}{n!} \right]^{-1} = \frac{1}{9}$$

and

$$\pi_1 = \frac{1.6}{1!}\pi_0 = 0.176$$

and so the probability that particular teller is idle is $0.111 + 0.5(0.176) = 0.199$.

Important Note: general queueing models are not understood to the same extent as $M/M/1$ (and $M/M/c$ to a lesser extent), and their given performance measurements may only be approximate and highly-dependent on the specifics of the problem at hand.

For this reason, $M/M/c$ models are sometimes used even when their use is not supported by the data (the situation is not unlike the widespread use of the normal distribution in a variety of probability and statistics problems).

In numerous applications, the empirical distributions of arrivals and service times are nearly Poisson and exponential, respectively, so that the assumption is not entirely off the mark, but numerical simulations should not be eschewed when departures from the $M/M/c$ model are too pronounced.

24.6 Exercises

The *Borealian Aeronautic Security Agency* (BASA) runs pre-board screening of passengers and crew for all flights departing the nation's airfields. There are 4 Major Airfields:

- Auckland
- Chebucto
- Saint-François
- Queenston

The screening process (PBS) is structurally similar at each airfield:

1. Passengers arrive at the beginning of the main queue
2. Boarding passes may or may not be scanned at S_1
3. Passengers enter the main queue
4. Boarding passes are scanned at S_2
5. Passengers are directed to a server entry position
6. Passengers and carry-on luggage are screened by a server

Some factors influence the PBS wait time, including:

- schedule intensity of departing flights
- passenger volume on these flights
- number of servers and processing rates at a given airfield, etc.

There might also be:

- yearly, seasonal, time-of-day, day-of-week interaction effects (among others) depending on the airfield, the flight destination, etc.
- trend level shifts in the number of passengers, flights, destinations, etc.

Datasets: `20262030.csv`, `BASA_AUC_2028_912.csv`, `dat_F_sub.csv`, `dat_P_sub_c.csv`.

1. Build a data dictionary for the datasets
2. Explore and visualize the datasets
3. Perform a queueing model analysis to predict the wait times at each airfield for which you have data.

Use the CATSA case study to inform your analysis [2].

Chapter References

[1] R. Berry. *Queueing Theory and Applications*. 2nd. PWS/Kent Publishing, 2002.

[2] P. Boily and J. Schellinck. *Introduction to Quantitative Consulting*. Quadrangle/Data Action Lab, 2025.

[3] A.K. Erlang. 'The theory of probabilities and telephone conversations'. In: *Nyt Tidsskrift for Matematik B* (1909).

[4] D.G. Kendall. 'Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain'. In: *The Annals of Mathematical Statistics* 24.3 (1953), pp. 338–354. DOI: 10.1214/aoms/1177728975.

[5] L. Kleinrock. *Queueing Systems, Volume I*. Wiley-Interscience. Wiley, 1974.

[6] 'Management Science and the Gas Shortage'. In: *Interfaces* 4.4 (Aug. 1974), pp. 47–51.

[7] C. Newell. *Applications of Queueing Theory*. Ettore Majorana International Science Series. Springer Netherlands, 2013.

[8] S.M. Ross. *Introduction to Probability Models*. 11th ed. San Diego, CA, USA: Academic Press, 2014.

[9] W.L. Winston. *Operations Research: Applications and Algorithms*. Cengage Learning, 2022.

Bayesian Data Analysis 25

by **Patrick Boily** and **Ehssan Ghashim**

Bayesian analysis is sometimes maligned by data analysts, due in part to the perceived element of arbitrariness associated with the selection of a meaningful prior distribution for a specific problem and the (formerly formidable) difficulties involved with producing posterior distributions for all but the simplest situations.

On the other hand, it has been said that "while classical data analysts need a large bag of clever tricks to unleash on their data, Bayesians only ever really need one." With the advent of efficient numerical samplers, modern data analysts cannot shy away from adding the Bayesian arrow to their quiver.

In this chapter, we introduce the basic concepts underpinning Bayesian analysis, and we present a small number of examples that illustrate the strengths of the approach.

25.1 Plausible Reasoning

> "A decision was wise, even though it lead to disastrous consequences, if the evidence at hand indicated it was the best one to make; and a decision was foolish, even though it lead to the happiest possible consequences, if it was unreasonable to expect those consequences." Herodotus, in Antiquity

Consider the following scenario [9]: while walking down a deserted street at night, you hear a security alarm, look across the street, and see a store with a broken window, from which a person wearing a mask crawls out with a bag full of smart phones.

The natural reaction might be to conclude that the person crawling out of the store is stealing merchandise from the store.

It might be the natural reaction, but how do we actually come to this conclusion? It **cannot** come from a **logical deduction based on evidence**.[1]

1: Such as would be used in mathematical reasoning.

Indeed, the person crawling out of the store **could have been** its owner who, upon returning from a costume party, realized that they had misplaced their keys just as a passing truck was throwing a brick in the store window, triggering the security alarm. Perhaps the owner then went into the store to retrieve items before they could be stolen, which is when you happened unto the scene.

If A is true, then B is true A is true	If A is true, then B is true B is true
B is true	A is more plausible (why?)
If A is true, then B is true B is false	If A is true, then B is true A is false
A is false	B is less plausible (why?)

Table 25.1: Deductive (left) vs. inductive (right) syllogisms.

But while the original reasoning process is not **deductive**, it is at least **plausible**, which in the logical context is called **inductive**.

We might also want to use a **weaker version** of inductive reasoning: let us say that we know that when A is true, then B is more plausible, and we also know that B is true. Then, we conclude that A is more plausible.

In the scenario described at the start of the section, if "the person is a thief" (A is true), you would not be surprised to "see them crawling out of the store with a bag of phones" (B is plausible). As you do "see them crawling out of the store with a bag of phones" (B is true), you would therefore not be surprised to find out that "the person is a thief" (A is plausible).

In **deductive reasoning**, we work from a cause to possible consequences; in **inductive reasoning**, we work from observations to possible causes.

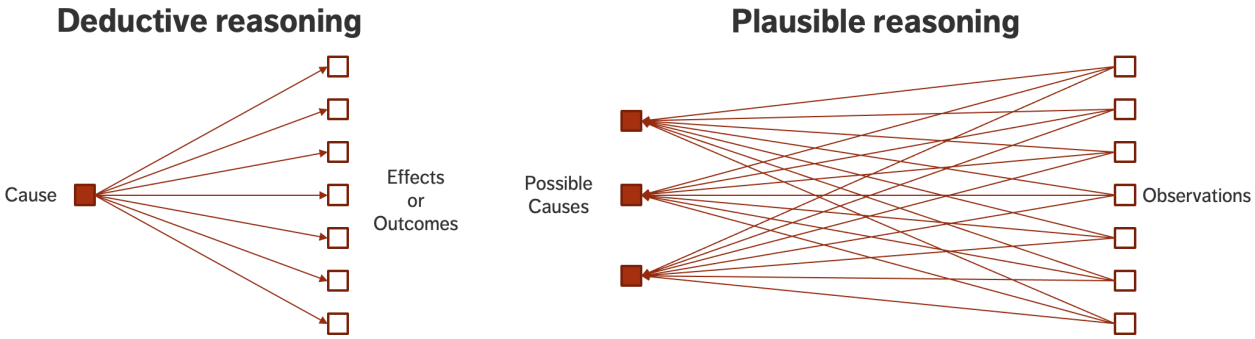


Figure 25.1: Deductive (left) vs. inductive (right) reasoning.

Plausibility relies on the notion of "surprise". In Tom Stoppard's 1966 play *Rosencrantz and Guildenstern are Dead* [15], Rosencrantz flips 92 heads in a row. This result is of course not impossible, but is it plausible? If this happened to you, what would you conclude?

25.1.1 Rules of Probability

Inductive reasoning requires methods to evaluate the validity of various propositions.

In 1763, Thomas Bayes [1] published a paper on the problem of induction, that is, on arguing from the specific to the general. In modern language and notation, Bayes wanted to use binomial data comprising r successes

out of n attempts to learn about the underlying chance $\boldsymbol{\theta}$ of each attempt succeeding. Bayes' key contribution was to use a probability distribution to represent uncertainty about $\boldsymbol{\theta}$. This distribution represents **epistemiological** uncertainty, due to lack of knowledge about the world, rather than **aleatory** (random) probability arising from the essential unpredictability of future events, as may be familiar from games of chance.

In this framework, a **probability** (plausibility) represents a 'degree-of-belief' about a proposition; the probability of an event will be recorded differently by two different observers, based on the respective background information to which they have access. This **Bayesian position** was the commonplace view of probabilities in the late 1700s and early 1800s, a view shared by such luminaries as Bernoulli and Laplace.[2]

Subsequent scholars found this vague and subjective,[3] and they redefined the probability of an event as its **long-run relative frequency**, given infinite repeated trials (the so-called **frequentist position**).

A forecast calling for rain with 90% probability doesn't mean the same thing to Bayesians and frequentists:

- in the Bayesian framework, this means that the forecaster is 90% certain that it will rain on the next day, say;
- in the frequentist framework, this means that it will rain on 90% of the days for which the model gives this forecast, in the long run.

The Bayesians framework is more aligned with how humans understand probabilities,[4] but how can we be certain that the **degree-of-belief** is a well-defined concept?

As it happens, there is a well-defined way to determine the rules of probability, based on a small list of axioms [3, 9]:

1. if a conclusion can be reasoned out in more than one way, then every possible way must lead to the same result;
2. all (known) evidence relevant to a question must be taken into consideration;
3. equivalent states of knowledge must be assigned the same probabilities;
4. if we specify how much we believe something is true, we have implicitly specified how much we believe it's false, and
5. if we have specified our degree-of-belief in a first proposition, and then our degree-of-belief in a second proposition if we assume the first one is true, then we have implicitly specified our simultaneous degree-of-belief in both propositions being true.

In what follows, we let I denote relevant background information; X, Y, and Y_k denote various propositions, and $-X$ or $\overline{X}$ denote the negation of proposition X.

The **plausibility** of X given I is denoted by $P(X \mid I)$; it is a real number whose value can range from 0 (**false**) to 1 (**true**). The rules of probability are quite simple:

- **Sum Rule:** for all propositions X, $P(X \mid I) + P(-X \mid I) = 1$;
- **Product Rule:** for all X, Y, $P(X, Y \mid I) = P(X \mid Y; I) \times P(Y \mid I)$.

From these two rules, we can also derive two useful corollaries:

2: Modern Bayesian statistics is still based on formulating probability distributions to express uncertainty about unknown quantities. These can be underlying parameters of a system (induction) or future observations (prediction). **Bayesian statistics** is a system for describing epistemiological uncertainty using the mathematical language of probability; **Bayesian inference** is the process of fitting a probability model to a set of data and summarizing the result with a probability distribution on the parameters of the model and on unobserved quantities (such as predictions).

3: How can you be sure that my degree-of-belief matches yours?

4: 92 heads in a row must mean that that the coin is biased, right?

- **Bayes' Theorem:** $P(X \mid Y; I) \times P(Y \mid I) = P(Y \mid X; I) \times P(X \mid I)$ (see next section);
- **Marginalization Rule:** $P(X \mid I) = \sum_k P(X, Y_k \mid I)$, where $\{Y_k\}$ are exhaustive and disjoint.[5]

5: Which is to say, $\sum_k P(Y_k \mid I) = 1$ and $P(Y_j, Y_k \mid I) = 0$ for all $j \neq k$).

For continuous variables, the marginalization rule becomes

$$P(X \mid I) = \int_{\Omega(Y)} P(X, Y \mid I)\, dY.$$

The **conditional probability** of A given B, $P(A \mid B)$ is the probability of A taking place given that another event B has occurred:

$$P(A \mid B; I) = \frac{P(A, B \mid I)}{P(B \mid I)} = \frac{P(A \cap B \mid I)}{P(B \mid I)}.$$

The probability that two events A and B both occur simultaneously is obtained by applying the **multiplication rule**:

$$P(A, B \mid I) = P(B \mid I) \times P(A \mid B; I) = P(A \mid I) \times P(B \mid A; I),$$

which we recognize as **Bayes' Rule**.

Classical Example: a family has two puppies that are not twins. What is the probability that the youngest puppy is female given that at least one of the puppies is female?[6]

6: Assume that male and female puppies are equally likely to be born.

Solution: our answer to this question follows a frequentist approach – we generate trials and identify successful events. There are 4 possibilities:

$$\{\text{MM}, \text{MF}, \text{FM}, \text{FF}\}.$$

Let A and B be the events that the youngest puppy is female and that at least one puppy is female, respectively; then

$$A \mid I = \{\text{FF}, \text{MF}\} \quad \text{and} \quad B \mid I = \{\text{FF}, \text{MF}, \text{FM}\},$$
$$\implies P(A \mid B; I) = \frac{P(A \cap B \mid I)}{P(B \mid I)} = \frac{2/4}{3/4} = 2/3.$$

25.1.2 Bayes' Theorem

Bayes' Theorem provides an expression for the conditional probability of A given B, that is:

$$P(A \mid B; I) = \frac{P(B \mid A; I) \times P(A \mid I)}{P(B \mid I)}$$
$$= \frac{P(B \mid A; I) \times P(A \mid I)}{P(B \mid A; I) \times P(A \mid I) + P(B \mid -A; I) \times P(-A \mid I)},$$

which is a direct application of the **Law of Total Probability**.

Bayes' Theorem can be thought of as a way of **coherently updating our uncertainty in the light of new evidence**. The use of a probability distribution as a 'language' to express our uncertainty is not an arbitrary choice: it can in fact be determined from deeper principles of logical reasoning or rational behaviour.

Example: consider a medical clinic (in what follows, we drop the explicit dependence on I to lighten the notation, but it is important to remember that it is there nonetheless).

- A could represent the event "Patient has liver disease." Past data suggests that 10% of patients entering the clinic have liver disease: $P(A) = 0.10$.
- B could represent the litmus test "Patient is alcoholic." Perhaps 5% of the clinic's patients are alcoholics: $P(B) = 0.05$.
- $B \mid A$ could represent the scenario that a patient is alcoholic, given that they have liver disease: perhaps we have $P(B \mid A) = 0.07$, say.

According to Bayes' Theorem, then, the probability that a patient has liver disease assuming that they are alcoholic is

$$P(A \mid B) = \frac{0.07 \times 0.10}{0.05} = 0.14$$

While this is a (large) increase over the original 10% suggested by past data, it remains unlikely that any particular patient has liver disease.

Bayes' Theorem with Multiple Events Let D represent some observed data and let A, B, and C be mutually exclusive (and exhaustive) events conditional on D. Note that

$$\begin{aligned} P(D) &= P(A \cap D) + P(B \cap D) + P(C \cap D) \\ &= P(D \mid A)P(A) + P(D \mid B)P(B) + P(D \mid C)P(C). \end{aligned}$$

According to Bayes' theorem,

$$\begin{aligned} P(A \mid D) &= \frac{P(D \mid A)P(A)}{P(D)} \\ &= \frac{P(D \mid A)P(A)}{P(D \mid A)P(A) + P(D \mid B)P(B) + P(D \mid C)P(C)}. \end{aligned}$$

In general, if there are n exhaustive and mutually exclusive outcomes $A_1, \ldots, A_n$, we have, for any $i \in \{1, \ldots, n\}$:

$$P(A_i \mid D) = \frac{P(A_i)P(D \mid A_i)}{\sum_{k=1}^{n} P(A_k)P(D \mid A_k)}$$

The denominator is simply $P(D)$, the **marginal distribution** of the data.

Note that, if the values of A_i are portions of the continuous real line, the sum may be replaced by an integral.

Example: In the 1996 General Social Survey, for males (age 30+):

- 11% of those in the lowest income quartile were college graduates.
- 19% of those in the second-lowest income quartile were college graduates.
- 31% of those in the third-lowest income quartile were college graduates.
- 53% of those in the highest income quartile were college graduates.

What is the probability that a college graduate falls in the lowest income quartile?

Solution: let Q_i represent the income quartiles ($P(Q_i) = 0.25$) and D represent the event that a male over 30 is a college graduate. Then

$$P(Q_1 \mid D) = \frac{P(D \mid Q_1)P(Q_1)}{\sum_{k=1}^{4} P(Q_k)P(D \mid Q_k)} = \frac{(0.11)(0.25)}{(0.11 + 0.19 + 0.31 + 0.53)(0.25)} = 0.09.$$

25.1.3 Bayesian Inference Basics

Bayesian statistical methods start with existing prior beliefs, and update these using data to provide posterior beliefs, which may be used as the basis for inferential decisions:

$$\underbrace{P(\boldsymbol{\theta} \mid D)}_{\text{posterior}} = \underbrace{P(\boldsymbol{\theta})}_{\text{prior}} \times \underbrace{P(D \mid \boldsymbol{\theta})}_{\text{likelihood}} / \underbrace{P(D)}_{\text{evidence}},$$

where the evidence is

$$P(D) = \int P(D \mid \boldsymbol{\theta})P(\boldsymbol{\theta})d\boldsymbol{\theta} \quad \text{or} \quad P(D) = \sum_k P(D \mid A_k)P(A_k),$$

where $\{A_k\}$ is mutually exclusive and exhaustive.

In the vernacular of Bayesian data analysis (BDA),

- the **prior**, $P(\boldsymbol{\theta})$, represents the strength of the belief in $\boldsymbol{\theta}$ without taking the observed data D into account;
- the **posterior**, $P(\boldsymbol{\theta} \mid D)$, represents the strength of our belief in $\boldsymbol{\theta}$ when the observed data D is taken into account;
- the **likelihood**, $P(D \mid \boldsymbol{\theta})$, is the probability that the observed data D would be generated by the model with parameter values $\boldsymbol{\theta}$, and
- the **evidence**, $P(D)$, is the probability of observing the data D according to the model, determined by summing (or integrating) across all possible parameter values and weighted by the strength of belief in those parameter values.

Central Data Analysis Question Bayes' Theorem allows is an essential component of the **scientific method** and knowledge discovery in general. Indeed, assume that an experiment has been conducted to determine the degree of validity of a particular hypothesis, and that corresponding experimental data has been collected.

The **central data analysis question** is the following: given everything that was known prior to the experiment, does the collected data support (or invalidate) the hypothesis?

Given everything that was known prior to the experiment, does the collected/observed data support (or invalidate) the hypothesis/presence of a certain condition?

The **problem** is that this is usually impossible to compute directly. Bayes' Theorem offers a **possible solution**:

$$\begin{aligned} P(\text{hypothesis} \mid \text{data}; I) &= \frac{P(\text{data} \mid \text{hypothesis}; I) \times P(\text{hypothesis} \mid I)}{P(\text{data} \mid I)} \\ &\propto P(\text{data} \mid \text{hypothesis}; I) \times P(\text{hypothesis} \mid I); \end{aligned}$$

the hope is that the terms on the right might be easier to compute than those on the left:

- $P(\text{hypothesis} \mid I)$ is the degree-of-belief that the hypothesis is true, **prior to the experiment**;
- $P(\text{hypothesis} \mid \text{data}; I)$ is the degree-of-belief that the hypothesis is true, **after the experimental data is taken into account**;
- $P(\text{data} \mid \text{hypothesis}; I)$ is the probability of observing experimental data, **assuming that the hypothesis is true**, and
- $P(\text{data} \mid I)$ is the probability of the experimental data being observed, **independently of the hypothesis**.

The theorem is often presented as

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}} \propto \text{likelihood} \times \text{prior},$$

i.e., **beliefs should be updated in the presence of new information**.

Example: "Most of us would have assigned almost no probability to terrorists crashing planes into buildings in Manhattan when we woke up on 9/11. But we recognized that a terror attack was an obvious possibility once the first hit the World Trade Center. And we had no doubt we were being attacked once the second tower was hit." [14]

Let A represent the proposition that a plane crashes into Manhattan skyscrapers. Let B represent the proposition that terrorists would attack Manhattan skyscrapers; before 2001, most people would only have assigned a miniscule probability to such an event, say 0.005%. There had been two incidents of planes crashing into Manhattan skyscrapers in the previous 25,000 days before September 11, 2001, so we might assign $P(A \mid -B; I) = 0.008\%$.

We could also assign a fairly high probability of a plane hitting a Manhattan skyscraper if terrorists were attacking said skyscrapers, say $P(A \mid B; I) = 95\%$.

PRIOR PROBABILITY		
Initial estimate of how likely it is that terrorists would crash planes into Manhattan skyscrapers	$P(B\|I) = x$	0.005%
A NEW EVENT OCCURS: FIRST PLANE HITS WTC		
Probability of plane hitting if terrorists are attacking Manhattan skyscrapers	$P(A\|B, I) = y$	95%+
Probability of plane hitting if terrorists are *not* attacking Manhattan skyscrapers (i.e., accident)	$P(A\|\bar{B}, I) = w$	0.008%*
POSTERIOR PROBABILITY		
Revised estimate of probability of terror attack, given first plane hitting WTC	$P(B\|A, I) = \frac{yx}{yx + w\,(1-x)}$	37%+

After one plane hitting the World Trade Center, our revised estimate of the probability of a terror attack now stands at roughly 37%. If a second plane hits the World Trade Center shortly after the first one, the posterior probability of a terror attack now jumps to a whopping 99.99%.

Determining an appropriate **prior** is a source of considerable controversy. Conservative estimates (**uninformative priors**) often lead to reasonable

PRIOR PROBABILITY		
Revised estimate of probability of terror attack (now that we know about the first plane hitting WTC)	$P(B\|I) = x^{\#}$	37%+
A NEW EVENT OCCURS: SECOND PLANE HITS WTC		
Probability of plane hitting if terrorists are attacking Manhattan skyscrapers	$P(A\|B,I) = y$	95%+
Probability of plane hitting if terrorists are *not* attacking Manhattan skyscrapers (i.e., accident)	$P(A\|\bar{B},I) = w$	0.008%*
POSTERIOR PROBABILITY		
Revised estimate of probability of terror attack, given second plane hitting WTC	$P(B\|A,I) = \frac{yx^{\#}}{yx^{\#} + w\,(1-x^{\#})}$	99.99%+

results, but in the absence of relevant information, it might be preferable to use **maximum entropy priors** (see Section 25.3).

The **evidence** is harder to compute on theoretical grounds – evaluating the probability of observing data requires access to some model as part of I. Either that model was good, so there's no need for a new hypothesis, or that model was bad, so we dare not trust our computation.[7]

7: Thankfully, the evidence is rarely required on problems of **parameter estimation**: prior to the experiment, there are numerous competing hypotheses; while the priors and likelihoods will differ, the evidence will not, so it is not needed to differentiate the various hypotheses.

25.1.4 Bayesian Data Analysis

The main characteristic of Bayesian methods is their explicit use of probability for quantifying uncertainty in inferences based on statistical data analysis. The process of **Bayesian data analysis** (BDA) can be idealized by dividing it into the following 3 steps:

1. Setting up a full probability model (the **prior**) – a joint probability distribution for all observable and unobservable quantities in a problem. The model should be consistent with knowledge about the underlying scientific problem and the data collection process (when available).
2. Conditioning on observed data (**new data**) – calculating and interpreting the appropriate posterior distribution (i.e., the conditional probability distribution of the unobserved quantities of ultimate interest, given the observed data).
3. Evaluating the fit of the model and the implications of the resulting posterior distribution (the **posterior**) – how well does the model fit the data? Are the substantive conclusions reasonable? How sensitive are the results to the modeling assumptions made in step 1? Depending on the responses, one can alter or expand the model and repeat the 3 steps.

The essence of Bayesian methods consists in identifying the **prior beliefs** about what results are likely, and then updating those according to the **collected data**.

For example, if the current success rate of a gambling strategy is 5%, we may say that it's reasonably likely that a small strategy modification could further improve that rate by 5 percentage points, but that it is most likely that the change will have little effect, and that it is entirely unlikely that the success rate would shoot up to 30%.[8]

8: After all, it is only a small modification.

As the data comes in, we **update our beliefs**. If the incoming data points to an improvement in the success rate, we move our prior estimate of the effect upwards; the more data we collect, the more confident we are in the estimate of the effect and the further we can leave the prior behind.

The end result is called the **posterior** – a probability distribution describing the likely effect of the strategy.

25.2 Simple Examples

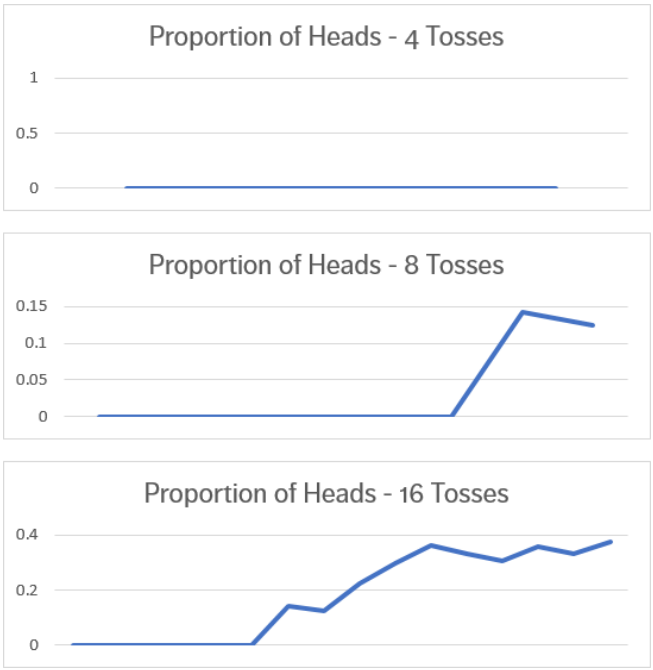
We take a look at three scenarios that will shed some light on the whole Bayesian entreprise:[9]

9: These examples will showcase how priors, likelihood, and posteriors interact.

- determining if a coin is fair (or not),
- finding a link between demographic information and salary, and
- estimating the number of dollar bills in circulation.

25.2.1 The Mysterious Coin

A mysterious stranger brings back a souvenir coin from a trip to a strange and distant land. They have been flipping it non-stop since their return. You can see the proportion of heads they obtained for 4, 8, and 16 tosses.



It might seem at first that the coin is biased, but the proportion of heads seems to inch its way towards 50%. What is truly going on?

Priors Perhaps the coin is not fair, coming as it does from a strange and distant land. Let us denote the coin's **bias** by H, i.e., the probability of flipping a head on a toss ($H \approx 0.5$: regular unbiased coins; $H \approx 0, 1$: highly biased coins). A **prior** for this scenario is a **probability density function** (p.d.f.)

$$P(\text{bias} = H) = P(H \mid I).$$

Four such priors are shown below.

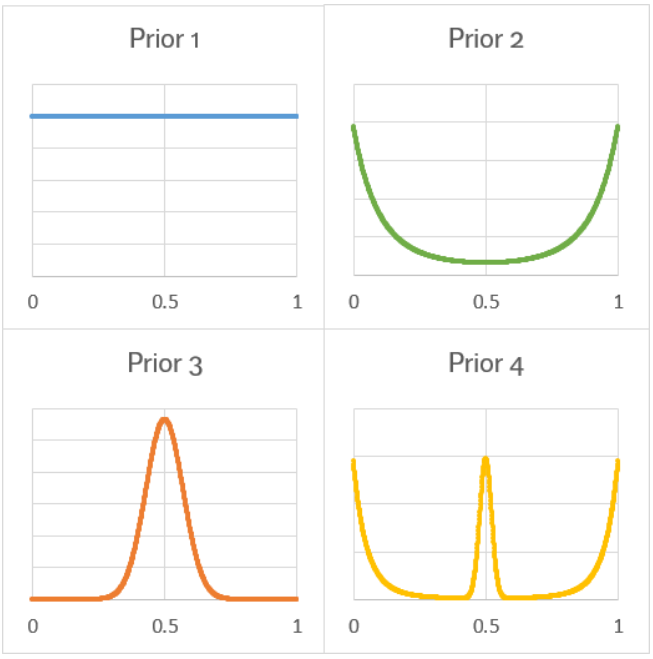


Figure 25.2: 4 priors for the fair coin problem: no idea (top left); suspect foul play (top right); just a regular coin (bottom left); probably just a regular coin, but the fact that somebody is even talking about this is suspicious (bottom right).

Why are we working with functions for the prior, when in the previous example (9/11 attacks), we only provided a number, $P(B \mid I) = 0.005\%$? In fact, we provided a **(discrete) function** as a prior:

$$P(B = x \mid I) = \begin{cases} 0.005\% & \text{if } x = \text{TRUE} \\ 99.995\% & \text{if } x = \text{FALSE} \end{cases}$$

Likelihood Let us assume that the coin has been tossed N times in total, and that K heads have been recorded. In this scenario, Bayes' Theorem takes the form:

$$P(\text{bias} = H \mid K \text{ heads}, N \text{ tosses}; I) \propto P(K \text{ heads}, N \text{ tosses} \mid \text{bias} = H; I) \times P(\text{bias} = H \mid I).$$

The **likelihood** is the probability of observing K heads in N tosses if the bias is H. If, as part of I, the tosses are independent (i.e., the result of one toss does not affect the others), then the likelihood is given by the binomial distribution

$$P(K \text{ heads}, N \text{ tosses} \mid \text{bias} = H; I) = \binom{N}{K} H^K (1 - H)^{N-K}.$$

Posteriors Combining the prior and the likelihood, we get:

$$P(\text{bias} = H \mid K \text{ heads}, N \text{ tosses}; I) \propto H^K(1-H)^{N-K} \times P_i(\text{bias} = H \mid I),$$

where i indexes the various prior scenarios described above.

We should thus be able to estimate the bias H^* by studying the posterior distribution for each of the 4 priors, for various number of throws N (see Figure 25.3):

- with the **non-committal prior** (blue p.d.f.)

  $$P_1(\text{bias} = H \mid I) \propto 1,$$

  the posterior is simply proportional to the likelihood; the central limit theorem seems to kick in after ≈ 30 tosses;
- with the **foul play prior** (green p.d.f.), we suspect early on that the bias is smaller than 0.5; the subsequent series of tosses moves the bias to a value $0.25 \leq H^* \leq 0.40$ quickly, as was the case with the non-informative prior – note the shrinking of the posterior with an increasing number of tosses;
- with the **regular coin prior** (orange p.d.f.)

  $$P_1(\text{bias} = H \mid I) \sim \mathcal{N}(0.5, \sigma^2),$$

  early results do not strongly suggest that the coin is biased (the prior gives little credence to the notion that the bias could lie in $0.25 \leq H^* \leq 0.40$), but the series of tosses forces the posterior to a biased distribution (note the smoother convergence of the posterior;
- with the **doubtful prior** (yellow p.d.f.), the competing hypotheses compete before converging to a bias, again in $0.25 \leq H^* \leq 0.40$. The convergence is more haphazard: as soon as one head or one tail is observed, the process nixes the two-sided coin option. Note the slower (and weirder) convergence to a gaussian posterior.

In the fair coin example, it would seem that the choice of a prior does not have much of an effect on the posterior ... **given enough data**.

This will not always be the case.

25.2.2 The Salary Question

Income information has been collected for 4782 individuals, together with demographic details: self-reported gender, age group, and education level (1 for post-secondary degree, 0 otherwise). The table below shows some of the summary statistics for the dataset; the dataset is available in `Salary.xlsx`.

Question: is there a link between demographic information and income? How would we answer this question using classical statistical methods? What if we had reason to suspect that reported incomes follow a (potentially) different distribution for each group? Would that change the approach?

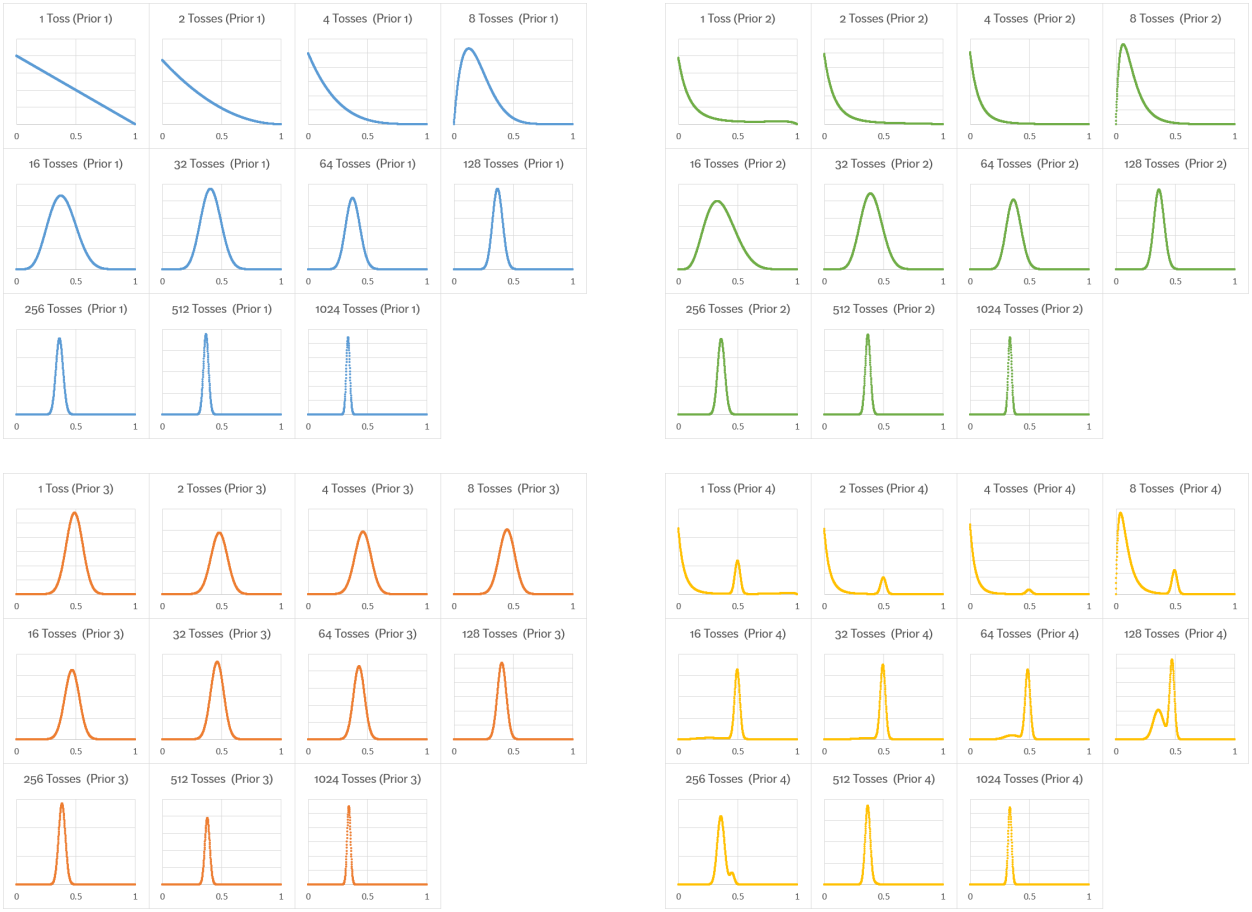


Figure 25.3: Posteriors for a different numbers of tosses; 4 priors, same data. After 128 tosses starting with the non-committal prior, we are fairly certain that the coin must be biased, with $0.25 \leq H^* \leq 0.40$ (top left); it takes roughly 256 tosses starting with the foul play prior for the same notion to arise (top right); after 512 tosses starting with the regular coin prior, we are fairly certain that the coin must be biased, with $0.33 \leq H^* \leq 0.40$ (bottom left); which is more or less the same when starting with the doubtful prior (bottom right).

Gender	Age	Edu	N	min	max
M	15-24	0	254	$ 13,970	$ 54,567
M	15-24	1	179	$ 25,871	$ 75,389
M	25-54	0	729	$ 15,560	$ 71,783
M	25-54	1	735	$ 28,329	$ 138,185
M	55-64	0	279	$ 21,966	$ 83,503
M	55-64	1	227	$ 58,384	$ 99,530
F	15-24	0	240	$ 917	$ 56,639
F	15-24	1	184	$ 20,361	$ 82,115
F	25-54	0	671	$ 14,161	$ 71,394
F	25-54	1	758	$ 22,691	$ 111,277
F	55-64	0	302	$ 20,719	$ 66,912
F	55-64	1	224	$ 53,840	$ 102,436
			4782		

In the Bayesian framework, we are interested in the posterior distribution

$$P(\text{parameters} \mid \text{data}; i, I), \quad i = 1, \ldots, 12.$$

If we assume (for no particular good reason) that the reported incomes are **normally distributed** for each group, then we seek

$$P(\mu_i, \sigma_i \mid \text{reported salaries in group } i; I), \quad i = 1, \ldots, 12.$$

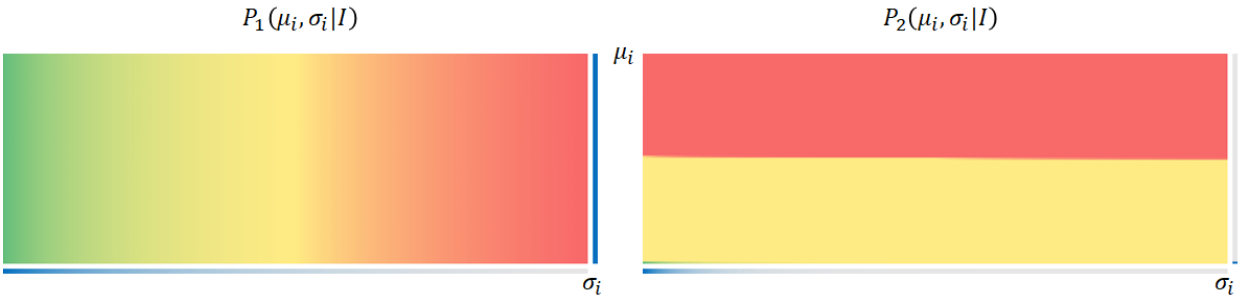


Figure 25.4: Two priors for the salary problem. Green represents large probabilities; red, low probabilities. The blue zones represent marginal probabilities of higher values.

Priors Determining a reasonable collection of **priors**

$$P(\mu_i, \sigma_i \mid I), \quad \text{for } i = 1, \ldots, 12,$$

is no easy task. One could naively pick a joint distribution which "**peaks**" at the sample mean $\overline{x}_i$, with standard deviation s_i, for each i, but there are sampling design issues associated with this approach.

Why not select, instead, a prior "which expresses **complete ignorance** except for the fact that μ_i is a **location** parameter and σ_i is a **scale** parameter" [9, 11]. This translates into using a **non-informative prior**

$$P_1(\mu_i, \sigma_i \mid I) \propto \sigma_i^{-1}, \quad i = 1, \ldots, 12$$

(we will discuss these further in the next section).

For comparison's sake, we will also consider the prior

$$P_2(\mu_i, \sigma_i \mid I) \propto \mu_i^{500} \sigma_i^{-4}, \quad i = 1, \ldots, 12.$$

The two priors are illustrated in Figure 25.4.

What could those priors represent, in the real world? What happens to the probabilities when σ_i increases? When μ_i increases? Note, as well, that these "priors" are not normalizable over the positive quadrant in (μ, σ)−space.[10]

10: The integral of these priors over the positive quadrant is infinite.

Instead, we could only consider them over a suitable finite sub-region; or use the fact that the product of the likelihood and the prior *is* normalizable.

Likelihood Let us denote the number of observations in group i by N_i. The likelihood is the probability

$$P(\text{reported incomes } \{x_{k,i}\} \text{ in group } i \mid \mu_i, \sigma_i; I), \quad i = 1, \ldots, 12.$$

We have assumed normality for any given observation. If we assume further that all observations are independent, then

$$P(\{x_{k,i}\} \mid \mu_i, \sigma_i; I) \propto \prod_{k=1}^{N_i} \sigma_i^{-1} \exp\left(\frac{-(\mu_i - x_{k,i})^2}{2\sigma_i^2}\right), \quad i = 1, \ldots, 12.$$

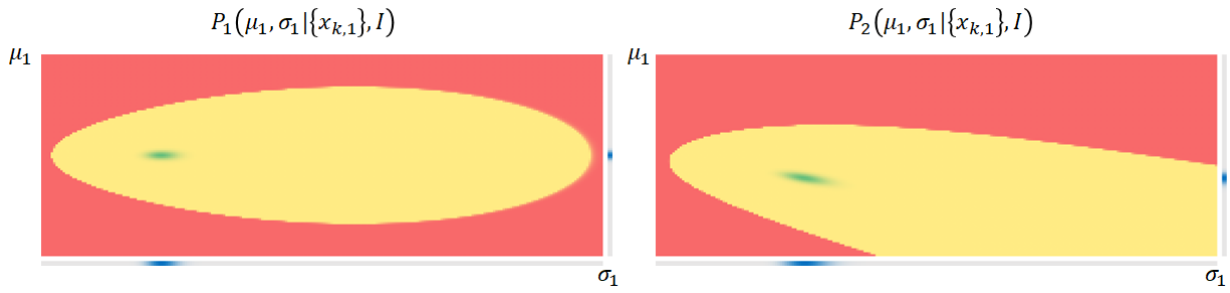


Figure 25.5: Posteriors for the salary problem (one per prior), for group $i = 1$. Green represents large probabilities; red, low probabilities. Note the shape of the posteriors. The blue zones represent marginal probabilities of higher values.

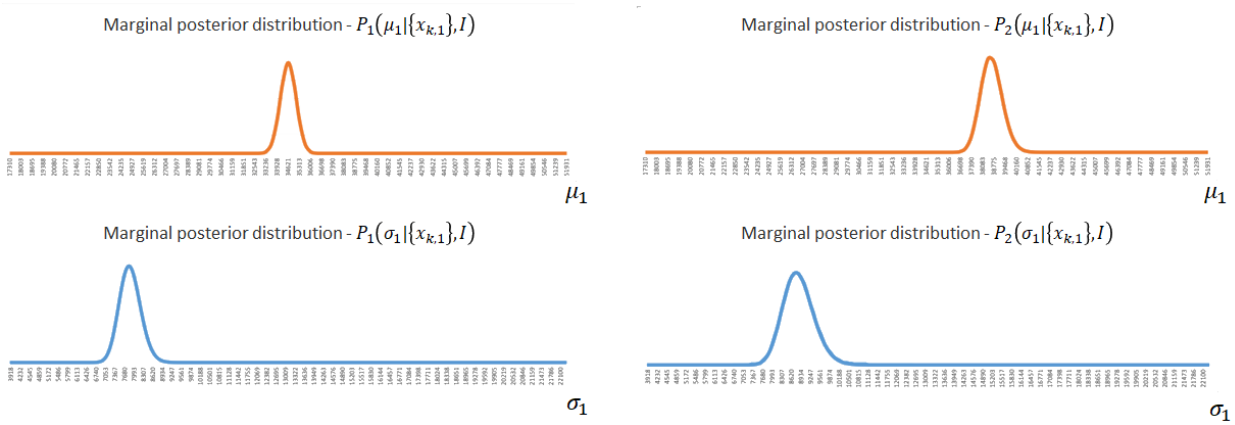


Figure 25.6: Marginal posteriors for the salary problem (one for each of the priors), for group $i = 1$. Note the differences in the distributions for each scenario.

Posteriors Combining the prior and the likelihood, we get, for the first prior:

$$P_1(\mu_i, \sigma_i \mid \{x_{k,i}\}; I) \propto \sigma_i^{-(N_i+1)} \prod_{k=1}^{N_i} \exp\left(\frac{-(\mu_i - x_{k,i})^2}{2\sigma_i^2}\right), \quad i = 1, \ldots, 12,$$

while for the second prior:

$$P_2(\mu_i, \sigma_i \mid \{x_{k,i}\}; I) \propto \mu_i^{500} \sigma_i^{-(N_i+4)} \prod_{k=1}^{N_i} \exp\left(\frac{-(\mu_i - x_{k,i})^2}{2\sigma_i^2}\right), \quad i = 1, \ldots, 12,$$

over some suitable sub-region in parameter space.

The joint posterior distributions for (μ_1, σ_1) (one for each of the priors) when $i = 1$ are shown in Figure 25.5.

We can read the likely values of each parameters for each scenario by looking at the spikes in the marginal posteriors of Figure 25.6.

The first two examples were (somehow ashamedly) conducted with Excel; the next example shows how we can use programmatical tools (like R) to answer questions using Bayesian analysis.

25.2.3 Money ($ Bill Y'All)

The **question:** how many 5$ dollar bills are there in circulation?

The **problem:** we cannot count them all – so what do we do?

The **solution:** "catch-and-release"

1. Capture a few 5$ bills.
2. Mark them and put them back in circulation.
3. At some later point, capture a few 5$ bills.
4. Count how many are marked.

$x = 500$ bills might have been **marked initially**, say; $y = 300$ bills might have been **re-captured** at stage 3, of which $w = 127$ were **marked**.

What is the most probable number of bills N in circulation?

Unlike the previous examples where we were trying to estimate the parameters from the data, we are trying to estimate data from parameters (**generative model**) – we do not compute the likelihood directly.

Simple Model In the simplest model, we might proceed as follows:

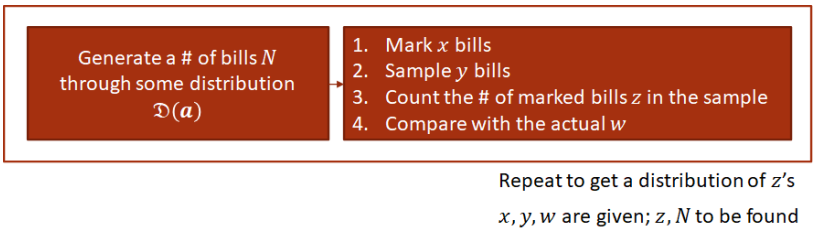


Figure 25.7: Catch-and-release schematics in the simple model.

1. We start by drawing a **large** random sample of # of bills N from an acceptable "prior" distribution on the parameters.
2. Using the Ns and the generative model (with x and y given – the observed values), we produce a (synthetic) # of marked bills z in each sample.
3. Finally, we only retain those values of N for which $z = w$.

Let us implement this in R using the values of x, y, and w provided above. We will generate priors using 500,000 replicates:

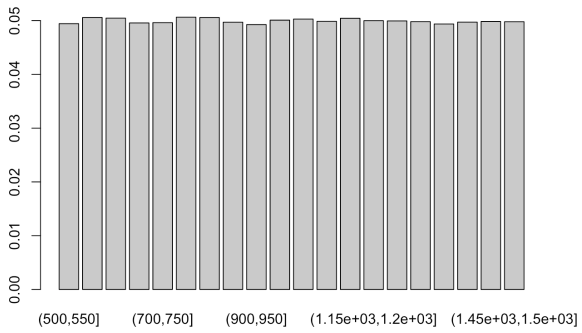
```
set.seed(1) # for replicability
N.draw = 500000    # number of replicates
x = 500 # number of bills marked in the initial capture
y = 300 # number of bills sampled in the second capture
w = 127 # number of marked bills in the second capture
```

Since $x = 500$ were first "captured", we know that there are at least 500 bills in circulation. To keep things from getting out of hands, we select a theoretical maximum for the number of bills in circulation.

```
upper.limit = 1500 # maximum (theoretical) number of bills
bin.width = 50     # for plotting the posterior
```

We now draw to create the prior distribution on the possible number of bills N_{bills} in circulation:

```
N.bills = sample (x:1500, N.draw, replace=TRUE)
barplot(table(cut(N.bills, seq(x, upper.limit, bin.width))) /
          length(N.bills), col = "gray")
```



A priori, all of these are "equally likely". Now, we use the observed "catch-and-release" data to define the generative model, in which we capture $x = 500$ bills in the first round, and $y = 300$ in the second:

```
pick.bills <- function(N.bills) {
  bills <- rep(0:1, c(N.bills - x, x)) # 0 for un-marked
                    # 1 for marked in the inital capture
  sum(sample(bills, y)) # sampling y bills in the 2nd round
}
```

The number of re-captured bills (for each trial) is simulated below:

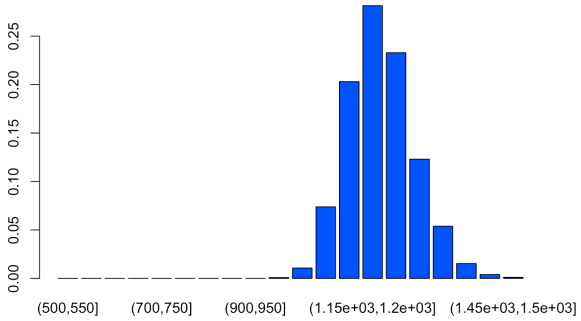
```
N.marked <- rep(NA, N.draw)
  for(i in 1:N.draw) {
    N.marked[i] <- pick.bills(N.bills[i])
  }
```

In the language of the generative model, `N.marked` is z. Now, we only keep those trials for which there were $w = 127$ re-captured marked bills, and retain the number of bills in circulation for these trials:

```
post.bills <- N.bills[N.marked == w]
```

Finally, we plot the posterior distribution:

```
barplot(table(cut(post.bills, seq(x,upper.limit,bin.width))) /
          length(post.bills), col = "blue")
```



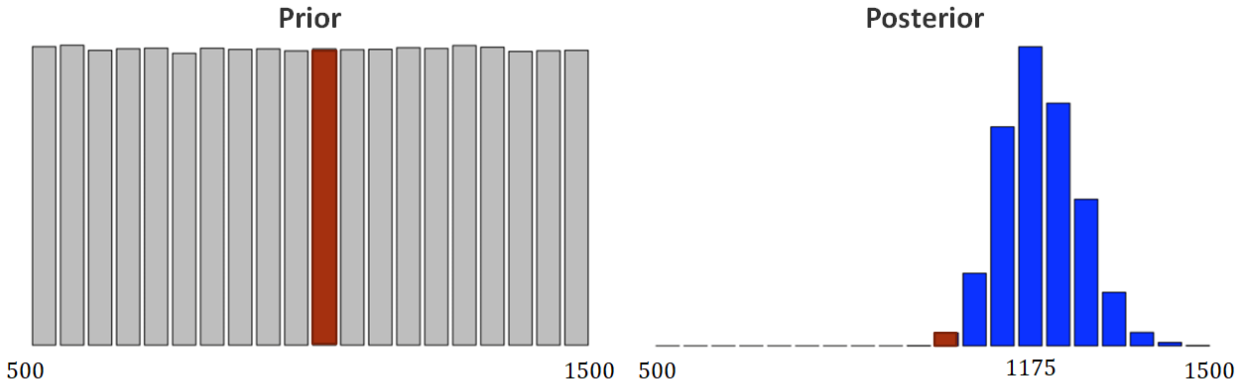
The summary statistics for the posterior distribution of the number of bills in circulation is thus:

```
length(post.bills)
summary(post.bills)
```

```
[1] 4754
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    979    1143    1188    1193    1236    1492
```

In other words, out of 500,000 trials, a little fewer than 5000 had the right characteristics (x, y, and w as observed in the "real world"), and the average/median number of bills in circulations for this smaller subset of trials is a tad below 1200. The Bayesian situation is illustrated below.[11]

11: We used a different seed, so the charts are slightly different, but the main ideas



$$P(N = 1000|z = 127, I) \propto P(z = 127|N = 1000, I) \times P(N = 1000|I)$$

Model: Marked Bills are Brittle It may be the case that the process of marking the bills might damage them somehow, so that they may be retired sooner than one would expect (with probability $u = 90\%$, say).

In this case, we might proceed as follows:

1. We start by drawing a **large** random sample of # of bills N from an acceptable "prior" distribution on the parameters.
2. Using the Ns and the generative model (with x, y, and u given – the observed values), we produce a (synthetic) # of marked bills z in each sample.
3. Finally, we only retain those values of N for which $z = w$.

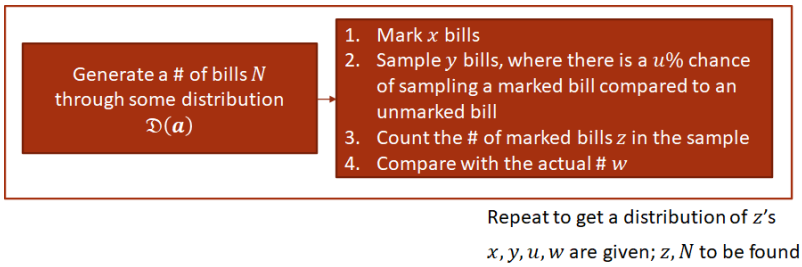
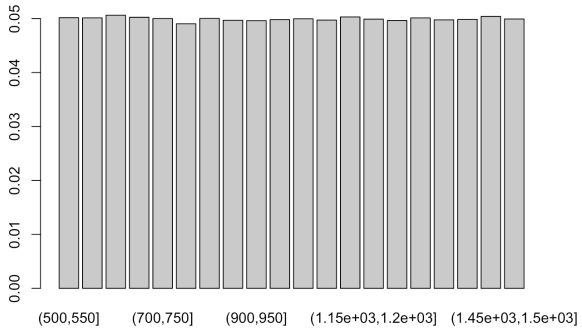


Figure 25.8: Catch-and-release schematics in the brittle model.

Let us implement this in R using the values of x, y, u, and w provided above. We will generate priors using 500,000 replicates:

```
set.seed(10) # for replicability
N.draw = 500000    # number of replicates
x = 500 # number of bills marked in the initial capture
y = 300 # number of bills sampled in the second capture
w = 127 # number of marked bills in the second capture
u = 0.9 # probability that marked bills will be retired
upper.limit = 1500 # maximum (theoretical) number of bills
bin.width = 50     # for plotting the posterior
N.bills = sample (x:1500, N.draw, replace=TRUE)
barplot(table(cut(N.bills, seq(x, upper.limit, bin.width))) /
          length(N.bills), col = "gray")
```



A priori, all of these are "equally likely" in the brittle scenario too. Now, we use the observed "catch-and-release" data to define the generative model, in which we capture $x = 500$ bills in the first round, and $y = 300$ in the second round, knowing that $u = 0.9$ of first round marked bills will be retired.[12]

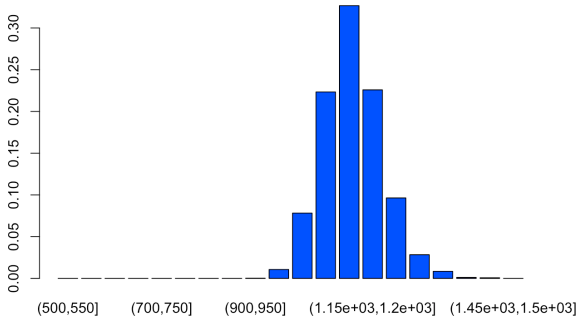
12: Would we expect there to be more bills in circulation, given these observations, in the brittle case or the simple case?

```
pick.bills <- function(N.bills) {
  bills <- rep(0:1, c(N.bills - x, x))
  prob.pick <- ifelse(bills == 0, 1.0, u) # brittleness
  sum(sample(bills, y, prob = prob.pick))
}
```

The number of re-captured bills (for each trial) is simulated below:

```
N.marked <- rep(NA, N.draw)
  for(i in 1:N.draw) {
```

```
    N.marked[i] <- pick.bills(N.bills[i])
  }

# Posterior distribution
post.bills <- N.bills[N.marked == w]
barplot(table(cut(post.bills, seq(x,upper.limit,bin.width))) /
          length(post.bills), col = "blue")
```



The summary statistics for the posterior distribution of the number of bills in circulation is thus:

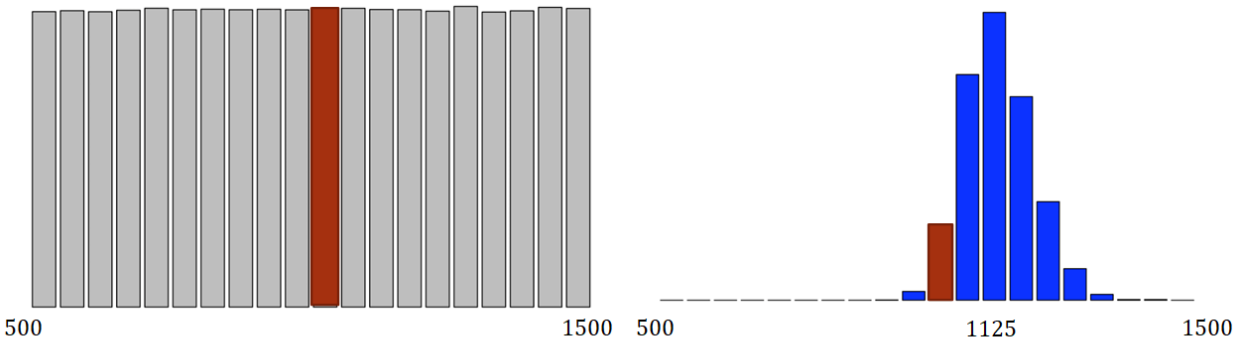
```
length(post.bills)
summary(post.bills)
```

```
[1] 4410
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    935    1089    1129    1132    1172    1411
```

In other words, out of 500,000 trials, about 4400 had the right characteristics (x, y, u, and w as observed in the "real world"), and the average/median number of bills in circulations for this smaller subset of trials is a roughly 1130. Does this make sense, given the brittleness assumption?

The Bayesian situation is illustrated below.[13]

13: Again, the charts are slightly different due the use of a different seed.



$$P(N = 1000|z = 127, I) \propto P(z = 127|N = 1000, I) \times P(N = 1000|I)$$

Model: Listen to the Banker Let us say that an old banker thinks that there should be about 1000 bills in circulation. How can we incorporate this piece of information?

In this case, we might proceed as follows:

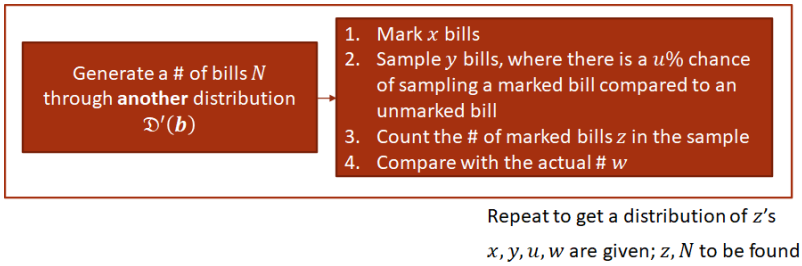


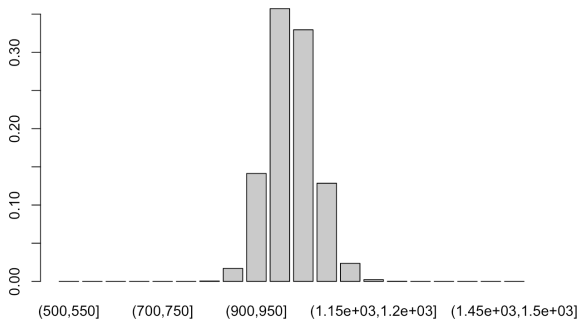
Figure 25.9: Catch-and-release schematics in the expert model.

Let us implement this in R using the values of x, y, u, and w provided above, as well as the expert's best guess. We will generate priors using 500,000 replicates:

```
set.seed(100) # for replicability
N.draw = 500000 # number of replicates
x = 500 # number of bills marked in the initial capture
y = 300 # number of bills sampled in the second capture
w = 127 # number of marked bills in the sample
u = 0.9 # probability that marked bills will be retired
banker.mean = 1000 # banker guess
upper.limit = 1500 # maximum (theoretical) number of bills
bin.width = 50     # for plotting the posterior
```

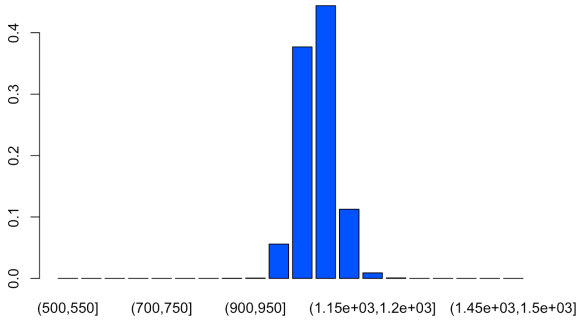
We now draw to create the prior distribution on the possible number of bills N_{bills} in circulation, using the banker's experience (instead of a uniform distribution, the prior might follow a binomial distribution with mean 1000$, say).

```
N.bills = rnbinom(N.draw, mu = banker.mean - x, size = w) + x
barplot(table(cut(N.bills, seq(x, upper.limit, bin.width))) /
          length(N.bills), col = "gray")

pick.bills <- function(N.bills) {
  bills <- rep(0:1, c(N.bills - x, x))
  prob.pick <- ifelse(bills == 0, 1.0, u)
  sum(sample(bills, y, prob = prob.pick))  second capture
}
```



The number of re-captured bills (for each trial) is simulated below:

```
N.marked <- rep(NA, N.draw)
  for(i in 1:N.draw) {
    N.marked[i] <- pick.bills(N.bills[i])
  }

# Posterior
post.bills <- N.bills[N.marked == w]
barplot(table(cut(post.bills, seq(x,upper.limit,bin.width))) /
          length(post.bills), col = "blue")
```

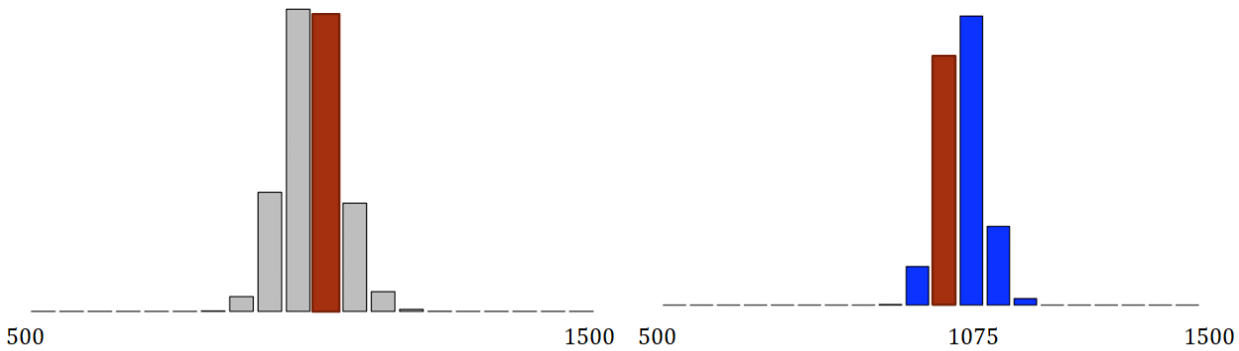


The summary statistics for the posterior distribution of the number of bills in circulation is thus:

```
length(post.bills)
summary(post.bills)
```

```
[1] 5258
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    893    1031    1057    1058    1083    1209
```

In other words, out of 500,000 trials, about 5250 had the right characteristics (x, y, u, and w as observed in the "real world"), and the average/median number of bills in circulations for this smaller subset of trials is a roughly 1050. Does this make sense, given the banker's opinion and the observations? The Bayesian situation is illustrated below.



$$P(N = 1000|z = 127, I) \propto P(z = 127|N = 1000, I) \times P(N = 1000|I)$$

25.3 Prior Distributions

Specifying a model means, by necessity, providing a prior distribution for the unknown parameters $\boldsymbol{\theta}$. The prior plays a critical role in Bayesian inference through the updating statement:

$$P(\boldsymbol{\theta} \mid D) \propto P(\boldsymbol{\theta}) \times P(D \mid \boldsymbol{\theta}).$$

In the Bayesian approach, all unknown quantities are described probabilistically, even before the data has been observed.

All priors are **subjective** in the sense that the decision to use a prior is left completely up to the researcher. But the choice of priors **is no more subjective than the choice of likelihood, the selection or collection of a sample, the estimation, or the statistic used for data reduction**. The choice of a prior can substantially affect posterior conclusions, however, especially with small sample sizes.

25.3.1 Conjugate Priors

The main challenge of Bayesian methods is that the posterior distribution of the vector $\boldsymbol{\theta}$ might not have an analytical form. Specifically, producing marginal posterior distributions from high-dimensional posteriors by repeated analytical integration may be difficult or even impossible mathematically.

There are exceptions however, providing easily obtainable computational posteriors through the use of a **conjugate prior**. Conjugacy is a joint property of a prior and a likelihood implying that the posterior has the same distributional form as the prior, but with different parameter(s).

The table below represents some common likelihoods and their conjugate priors (an extensive list can be found in [16]).

Likelihood	Prior	Hyperparameters
Bernoulli	Beta	$\alpha > 0, \beta > 0$
Binomial	Beta	$\alpha > 0, \beta > 0$
Poisson	Gamma	$\alpha > 0, \beta > 0$
Normal for μ	Normal	$\mu \in \mathbb{R}, \sigma^2 > 0$
Normal for σ^2	Inverse Gamma	$\alpha > 0, \beta > 0$
Exponential	Gamma	$\alpha > 0, \beta > 0$

For instance, if the probability of s successes in n trials (the **likelihood**) is given by

$$P(s, n \mid q) = \frac{n!}{s!(n-s)!} q^s (1-q)^{n-s}, \quad q \in [0, 1],$$

and the **prior probability** for q follows a Beta(α, β) distribution with $\alpha > 0, \beta > 0$, so that

$$P(q) = \frac{q^{\alpha-1}(1-q)^{\beta-1}}{B(\alpha, \beta)}, \quad \text{for } q \in [0, 1],$$

then the **posterior distribution** for q given s successes in n trials follows a Beta$(\alpha + s, \beta + n - s)$ distribution, so that

$$P(q \mid s, n) = \frac{P(s, n \mid q) \times P(q)}{P(s, n)} = \frac{q^{\alpha+s-1}(1-q)^{\beta+n-s-1}}{B(\alpha + s, \beta + n - s)}, \quad \text{for } q \in [0, 1].$$

Conjugate priors are mathematically convenient, and they can be quite flexible, depending on the specific hyperparameters we use; but **they reflect very specific prior knowledge and should be eschewed unless we truly possess that prior knowledge**.

25.3.2 Uninformative Priors

An **uninformative prior** (or **objective** prior) is one which intentionally provides very little specific information about the parameters of interest. Uninformative priors are very useful from the perspective of traditional Bayesianism seeking to mitigate the frequentist criticism of **intentional subjectivity**.

The rationale for using uninformative prior distributions is often said to be 'to let the data speak for itself,' so that inferences are unaffected by information external to the current data.

A classic uninformative prior is the **uniform prior**. A proper uniform prior integrates to a finite quantity and is thus normalizable. For example, for data following a Bernoulli(θ) distribution, a uniform prior on θ is

$$P(\theta) = 1, \quad 0 \leq \theta \leq 1.$$

For data following a $N(\mu, 1)$ distribution, say,[14] the uniform prior on the support of μ is improper as

$$P(\mu) = 1, \quad -\infty < \mu < \infty$$

diverges; however, such a choice could still be acceptable as long as the resulting posterior is normalizable.[15] As there are instances where an improper prior yields an improper posterior, care is warranted.

This is also called the **principle of indifference**, which states that with no evidence one way or another, degrees of belief should be distributed equally among all the considered outcomes.[16]

There are plenty of situations where the uniform prior is **not** an appropriate prior; such a prior makes assumptions about the distribution of the parameters of interest that fall squarely in the **subjective camp**. The use of uniform priors is often justified solely on the basis of convenience.[17]

The **Jeffreys prior** is an approach to generate uninformative priors. For a given random parameter $\boldsymbol{\theta}$, the Jeffreys prior is

$$P(\boldsymbol{\theta} \mid I) \propto \sqrt{\det \mathcal{I}(\boldsymbol{\theta})},$$

where $\mathcal{I}(\boldsymbol{\theta})$ represents the **Fisher information**, which measures the amount of information that an observable random variable X implies about an unknown parameter vector $\boldsymbol{\theta}$ (i.e., we are interested in $P(X \mid \boldsymbol{\theta})$).

14: Or any data with **unbounded support**.

15: Which is to say, the integral of the posterior converges on its support.

16: But *Bertrand's Paradox* provides doubt as to the validity of this principle.

17: Since the posterior is then simply proportional to the likelihood.

Let $f(X \mid \boldsymbol{\theta})$ be the corresponding p.d.f./p.m.f.;

$$\left[\mathcal{I}(\boldsymbol{\theta})\right]_{i,j} = -\,\mathrm{E}\left[\left.\frac{\partial^2}{\partial\theta_i\,\partial\theta_j}\log f(X \mid \boldsymbol{\theta})\right| \boldsymbol{\theta}\right].$$

Note that the Jeffreys prior depends on underlying statitistical model:

- if X follows a normal distribution $\mathcal{N}(\mu, \sigma^2)$, with σ fixed, and all we assume is that μ is a **location** parameter, then the Jeffreys prior would be
$$P(\mu \mid I) \propto 1,$$
an improper uniform distribution (all locations are equally likely to be the mean);
- if X follows a normal distribution $\mathcal{N}(\mu, \sigma^2)$, with μ fixed, and all we assume is that $\sigma > 0$ is a **scale** parameter, then it would be
$$P(\sigma \mid I) \propto \frac{1}{\sigma},$$
again an improper distribution, but one for which a dispersion σ becomes progressively less likely as it increases;
- if X follows a Poisson distribution $\mathcal{P}(\lambda)$ and all we assume is that $\lambda \geq 0$, then it would be the improper distribution
$$P(\lambda \mid I) \propto \frac{1}{\sqrt{\lambda}}.$$

In contrast, a **weakly informative prior** is one for which only **partial information** about a variable is available; the choice of a uniform prior is often weakly informative.

We will discuss another uninformative approach, the **Maximum Entropy prior**, shortly.

25.3.3 Informative Priors

Informative priors are those that **deliberately** insert information that researchers have at hand. This seems like a reasonable approach since previous scientific knowledge should play a role in statistical inference.

However, there are two important requirements for researchers:

1. the **overt** declaration of prior specification, and
2. a detailed sensitivity analysis to show the effect of these priors relative to uninformed types.

Transparency is required to avoid the common pitfall of **data fishing**; sensitivity analysis can provide a sense of exactly how informative the prior is. But where do informative priors come from, in the first place?

Generally these priors are derived from:

- past studies, published work, researcher intuition;
- interviewing domain experts;
- convenience with conjugacy, and
- non-parametric and other data-derived sources.

Prior information from past studies need not be in agreement. One useful strategy is to construct prior specifications from **competing school-of-thoughts** in order to contrast the resulting posteriors and produce informed statements about the relative strength of each of them.

Example: we have noted previously that a Bernoulli likelihood and a Beta prior form a set of conjugate priors. For this exercise, we use the `R` function `BernBeta()` defined in the excellent [10].[18]

1. Start with a prior distribution that expresses some uncertainty that a coin is fair: Beta($\theta \mid 4, 4$). Flip the coin once; assume that a Head is obtained. What is the posterior distribution of the uncertainty in the coin's fairness θ?

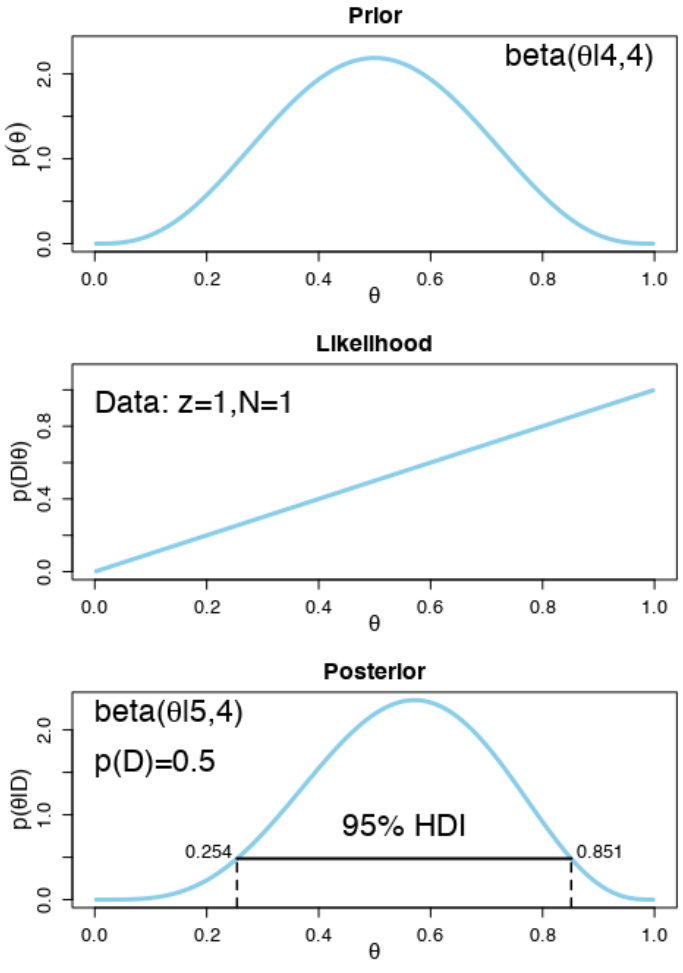
   Solution: we know, on theoretical grounds, that the posterior follows a

   $$\text{Beta}(\theta \mid 4+1, 4+1-1; I) = \text{Beta}(\theta \mid 5, 4; I)$$

   distribution.[19]

```
post = BernBeta( c(4,4) , c(1) )
show(post)
```

```
[1] 5 4
```



18: This function uses the conjugacy between the Bernoulli (likelihood) and the Beta (prior) distributions to determine the posterior distribution Beta for the uncertainty in the fairness of the coin (1 represents a H(ead) on the flip, 0 a T(ail)). Note that the function returns the posterior beta values each time it is called, so returned values can be fed back into the prior in a subsequent function call.

19: The label on the $y-$axis of the posterior distribution provides the posterior parameters.

2. Use the posterior parameters from the previous flip as the prior for the next flip. Suppose we flip again and get a H. What is the new posterior on the uncertainty in the coin's fairness?

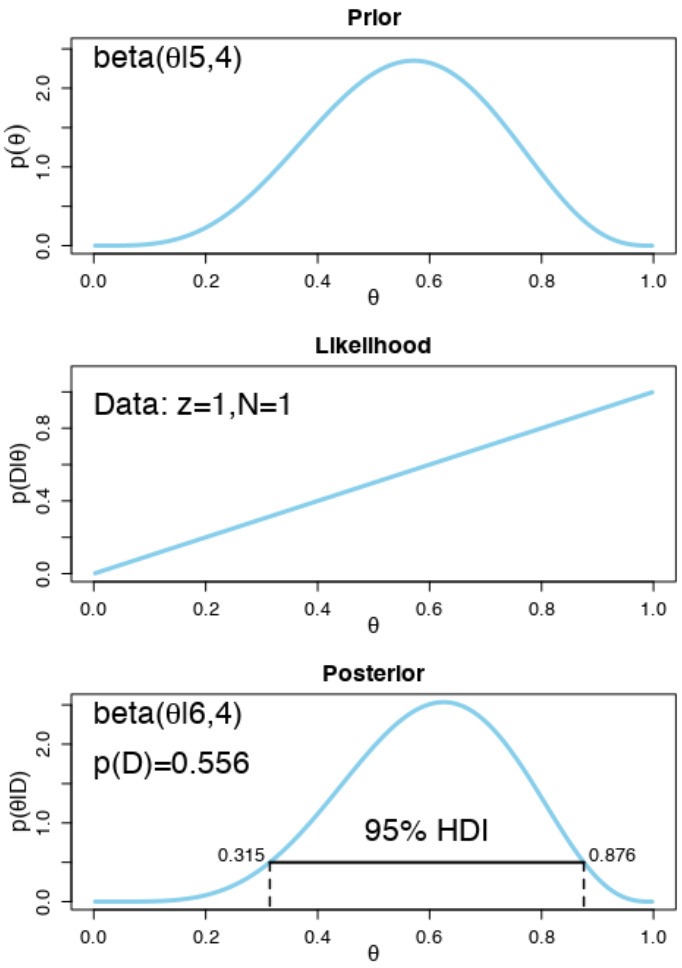
   Solution: on theoretical grounds, the posterior is

   $$\text{Beta}(\theta \mid 6, 4; I),$$

   which is shown below.

```
post = BernBeta( post , c(1) )
show(post)
```

```
[1] 6 4
```

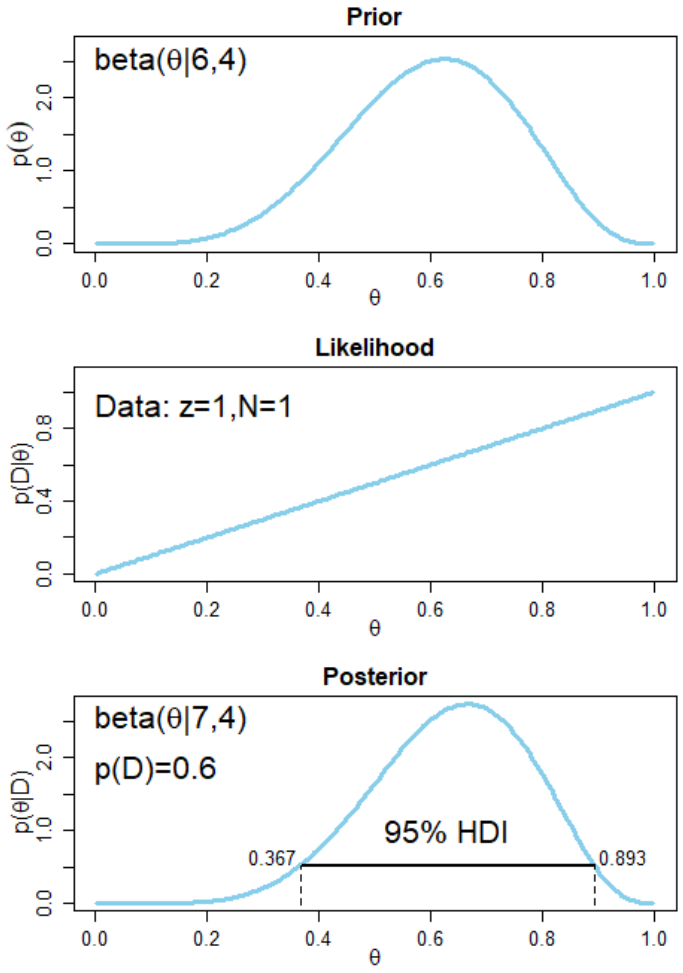


3. Using the most recent posterior as the prior for the next flip, flip a third time and obtain yet again a H. What is the new posterior?

   Solution: in this case, we know that the posterior for the coin's fairness follows a $\text{Beta}(\theta \mid 7, 4; I)$ distribution.

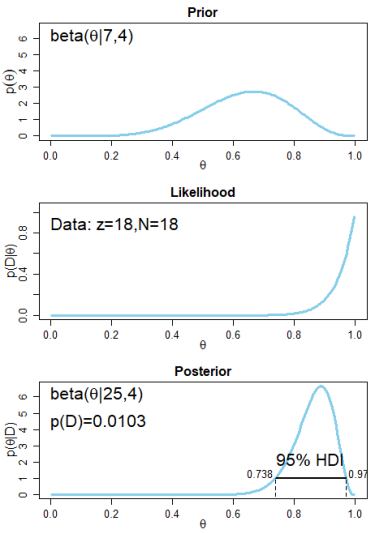
```
post = BernBeta( post , c(1) )
show(post)
```

```
[1] 7 4
```



> Should flipping 3 H in a row give us pause? Is there enough evidence to suggest that $\theta \neq 0.5$ (i.e, that the coin is not fair)? What if we were to flip 18 H in a row from this point on?[20]

20: The modified code would yield:



Note the posterior's bias.

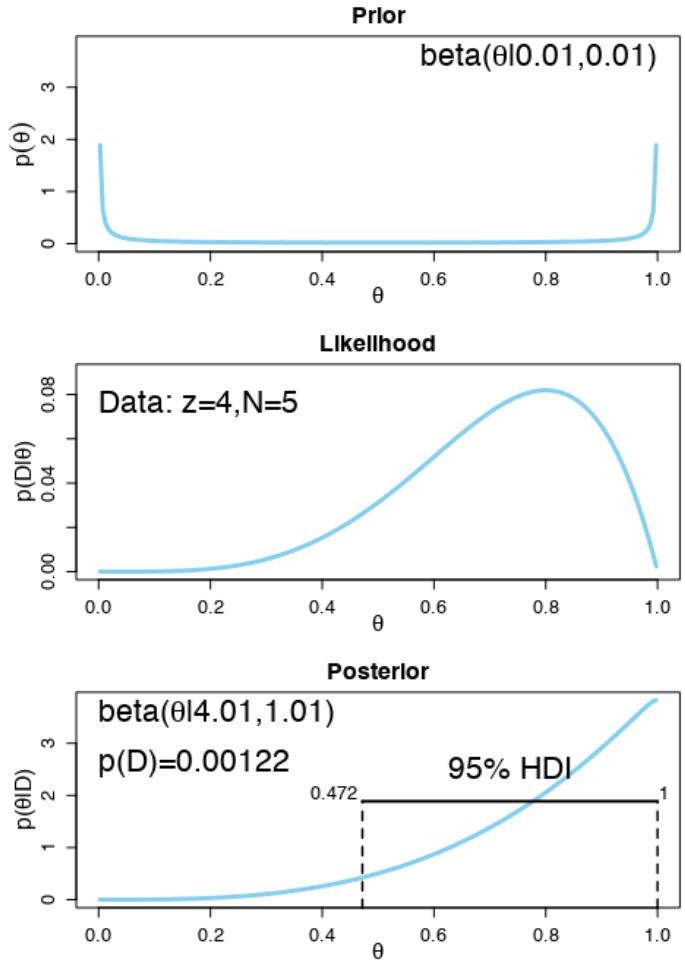
When working on a problem, it can be easy to get side-tracked and confused with the notation. In those cases, it is useful to return to the definition of each of the terms in Bayes' theorem (i.e., $P(\theta \mid D; I)$, $P(D \mid I)$, $P(D \mid \theta; I)$, etc.).

Example: suppose that a friend has a coin that we know comes from a magic store; as a result, we believe that the coin is strongly biased in either of the two directions (it could be a trick coin with both sides being H, for instance), but we don't know which one it favours. We will express the belief of this prior as a Beta distribution. Let's say that our friend flips the coin five times; resulting in 4 H and 1 T. What is the posterior distribution of the coin's fairness θ?

Solution: we start with a prior that corresponds with our assumptions, and assume 4 H and 1 T:

```
post = BernBeta( c(1,1)/100 , c(1,1,1,1,0) )
show(post)
```

```
[1] 4.01 1.01
```



This prior captures our belief that the coin is strongly biased, although we do not know in which direction the bias lies before seeing data. The use of 0.01 is arbitrary, in a sense; 0.1 would have worked as well, say.

The posterior distribution is

$$\text{Beta}(\theta \mid 4.01, 1.01; I),$$

which, as shown above, has its mode essentially at 1.0, and not near the mean ≈ 0.8. Is the coin indeed biased? In which direction?

How would the answer change if we had no reason to suspect that the coin was biased in the first place? These are all questions that could be answered by playing with `BernBeta()`.

25.3.4 Maximum Entropy Priors

Whether the priors are uninformative or informative, we search for the distribution that best encodes the prior state of knowledge from a set of trial distributions.

Consider a discrete space X of cardinality M with probability density $P(X) = \mathbf{p} = (p_1, ..., p_M)$. The **entropy** of such a $\mathbf{p}$, denoted by $H(\mathbf{p})$, is

given by

$$H(\mathbf{p}) = -\sum_{i=1}^{M} p_i \log p_i, \quad \text{with } 0 \cdot \log(0) = 0.$$

In the case of a continuous p.d.f. $P(\mathbf{X}) = P(X_1, \ldots, X_n)$ on some domain $\Omega \subseteq \mathbb{R}^n$, the entropy is given by

$$H(P) = -\int_{\Omega} P(\mathbf{Z}) \log(P(\mathbf{Z}))\, d\mathbf{Z}.$$

The **maximum entropy principle** (MaxEnt) states that, given a class of trial distributions with constraints, the optimal prior is the trial distribution with the largest entropy. As an example, the most basic constraint is for $\mathbf{p}$ to lie in the **probability simplex**, that is, $\sum_i p_i = 1$ and $p_i \geq 0$ for all i in the discrete case, or $\int_{\Omega} P(\mathbf{Z})\, d\mathbf{Z} = 1$ and $P(\mathbf{Z}) \geq 0$ on Ω in the continuous case.

Example: with no added constraint, the MaxEnt principle yields a prior which solves the optimization problem

$$\begin{aligned} \max \quad & -p_1 \log p_1 - \cdots - p_M \log p_M \\ \text{s.t.} \quad & p_1 + \cdots + p_M = 1 \text{ and } p_1, \ldots, p_M \geq 0. \end{aligned}$$

With the method of Lagrange multipliers, the optimization reduces to

$$\mathbf{p}^* = \arg_{\mathbf{p}} \max\{H(\mathbf{p}) - \lambda(p_1 + \cdots + p_M - 1)\},$$

whose solution is $\mathbf{p}^* \propto$ constant. Hence, subject to no additional constraints, the uniform distribution is the maximum entropy prior.

Example: we use Bayesian analysis to predict cab waiting times.

> "The joke about New York is that you can never get a cab, except when you don't need a cab, and then there are cabs everywhere" (quote and example from S.DeDeo's *Maximum Entropy Methods* tutorial [4]).

At various moments, we head out to the street to hail a cab, and we keep track of how long it took before a cab was available. Perhaps the observations (in minutes) look like this

$$6, 3, 4, 6, 2, 3, 2, 6, 4, 4.$$

What can you conclude about the waiting time for a New York cab?

Solution: in the best case scenario a cab is waiting for us as we get to the curb ($j = 0$), while in the worst case scenario (a zombie apocalypse, say?), no cab ever comes ($j \to \infty$). But can anything else be said?

To use MaxEnt in this situation, we need to find – among all of the trial distributions that could have generated the observed waiting times – the one with the highest entropy. Unfortunately, there are infinitely many such distributions.

We can narrow the search, however, by including a constraint stating that the expected value of the trial distributions should be the same as the mean of the sample: in this case, 4.

The two constraints translate to

$$g_1(\mathbf{p}) = \sum_{j=0}^{\infty} j \cdot p_j - 4 = 0 \quad \text{and} \quad g_2(\mathbf{p}) = \sum_{j=0}^{\infty} p_j - 1 = 0,$$

where p_j is the probability of having to wait j minutes for a cab.

The method of Lagrange multipliers reduces the problem to solving

$$\arg_{\mathbf{p}} \max \{H(\mathbf{p}) - \lambda_1 g_1(\mathbf{p}) - \lambda_2 g_2(\mathbf{p})\}.$$

This requires solving the gradient equation

$$\nabla_{\mathbf{p}} H(\mathbf{p}) = \lambda_1 \nabla_{\mathbf{p}} g_1(\mathbf{p}) + \lambda_2 \nabla_{\mathbf{p}} g_2(\mathbf{p}),$$

which gives rise to equations of the form

$$-(\ln p_j + 1) = \lambda_1 j + \lambda_2, \quad j = 0, 1, \ldots,$$

or simply $p_j = \exp(-\lambda_1 j)\exp(-1-\lambda_2)$ for $j = 0, 1, \ldots$.

Since

$$1 = \sum_{j=0}^{\infty} p_j = \exp(-1-\lambda_2) \sum_{j=0}^{\infty} \exp(-\lambda_1 j),$$

we have

$$\exp(1+\lambda_2) = \sum_{j=0}^{\infty} \exp(-\lambda_1 j) = \frac{1}{1-\exp(-\lambda_1)},$$

assuming that $|\exp(-\lambda_1)| < 1$.

Similarly,

$$4 = \sum_{j=0}^{\infty} j p_j = \exp(-1-\lambda_2) \sum_{j=0}^{\infty} j \exp(-\lambda_1 j),$$

so that

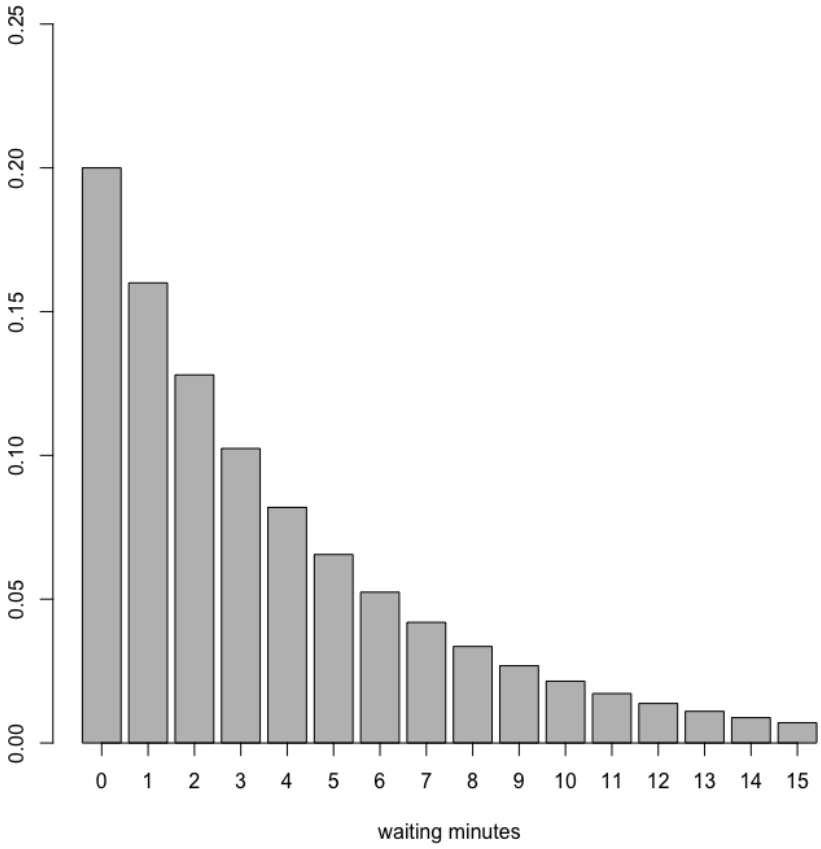
$$4\exp(1+\lambda_2) = \sum_{j=0}^{\infty} j \exp(-\lambda_1 j) = \frac{\exp(-\lambda_1)}{(1-\exp(-\lambda_1))^2}.$$

Substituting the first of these into the latter, and solving for λ_1, we see that $\lambda_1 = \ln(5/4)$. Substituting that result back into the first equation, we further obtain $\exp(-1-\lambda_2) = \frac{1}{5}$, so that

$$p_j = \exp(-1-\lambda_2)\exp(-\lambda_1 j) = \frac{1}{5}\left(\frac{4}{5}\right)^j, j = 0, \ldots$$

It is easy to see that this defines a distribution; a "verification" is provided by the following code.

```
pmf_maxent <- function(x,lambda=4/5) (1-lambda)*(lambda)^x
sum(pmf_maxent(0:100))  # check if it's a distribution
mp <- barplot(pmf_maxent(0:15), ylim=c(0,.25),
              xlab="waiting minutes")
axis(1,at=mp,labels=paste(0:15))
```



This distribution could be used as a prior in a Bayesian analysis of the situation. Notice that some information about the data (in this case, only the sample mean) is used to define the MaxEnt prior.

Crucially, however, the data that is used to build the MaxEnt prior **cannot** be re-used as part of the likelihood computations. The situation is not unlike that of the training/testing paradigm of machine learning.

25.4 Posterior Distributions

The posterior distribution is used to estimate a variety of **model parameters of interest**, such as the mean, the median, the mode, etc.

It is possible to construct **credible intervals/regions** directly from the posterior (in contrast to the "confidence" intervals of frequentist inference). Given a posterior distribution on a parameter $\theta \in \mathbb{R}$, a $1 - \alpha$ **credible region** C is a subset of $\mathbb{R}$ such that

$$P(\theta \in C \mid D; I) \geq 1 - \alpha.$$

A similar construction can be used for a joint credible region for $\boldsymbol{\theta} \in \mathbb{R}^n$.

Because the posterior is a full distribution on the parameters, it is possible to make all sorts of probabilistic statements about their values, such as:

- "I am 95% sure that the true parameter value is bigger than 0.5";
- "There is a 50% chance that θ_1 is larger than θ_2";
- etc.

25.4.1 High-Density Regions

We can build the credible interval of θ-values using the **highest density region** (HDR); i.e., we define a region $\mathbf{C}_k$ in parameter space with

$$\mathbf{C}_k = \{\theta \mid P(\theta \mid D; I) \geq k\},$$

where k is the largest number such that

$$\int_{\mathbf{C}_k} P(\theta \mid D; I)\, d\theta = 1 - \alpha.$$

This typically has the effect of finding the **smallest region** $\mathbf{C}_k$ (in measure) meeting the criterion.[21]

21: The value k can be thought of the height of a horizontal line (or hyperplane, in the case of multivariate posteriors) overlaid on the posterior, whose intersection(s) with the latter define a region over which the integral of the posterior is $1 - \alpha$. In most cases, it must be found numerically.

Example: it is an election year and we are interested in knowing whether the general population prefers candidate A or candidate B. A recently published poll states that of 400 randomly sampled voters, 232 preferred candidate A, while the remainder preferred candidate B.

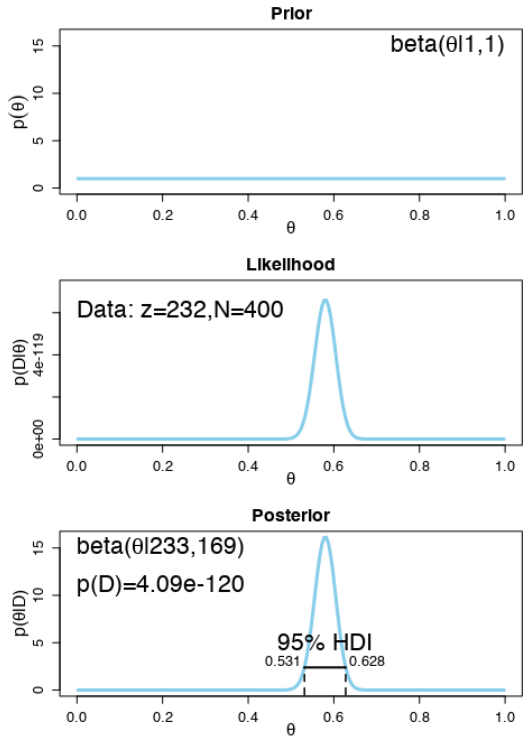
1. Suppose that we had no particular belief about the preference before the poll was published.[22] What is the 95% HDI on this belief after learning of the poll result?

   Solution: let preference for candidate A be denoted by 1, and preference for candidate B by 0. We can think of each voter's preference as arising from an independent Bernoulli trial.[23]

22: A non-informative uniform prior on the preference, which is to say, a Beta distribution with both parameters equal to 1.

23: Assuming that the polled voters are selected randomly.

```
post = BernBeta(c(1,1), c(rep(1,232), rep(0,168)))
```



We see that the posterior distribution's 95% HDI ranges from 0.531 to 0.628, in favour of candidate A.

2. Based on the poll, is it credible to believe that the population is equally divided in its preferences among candidates?

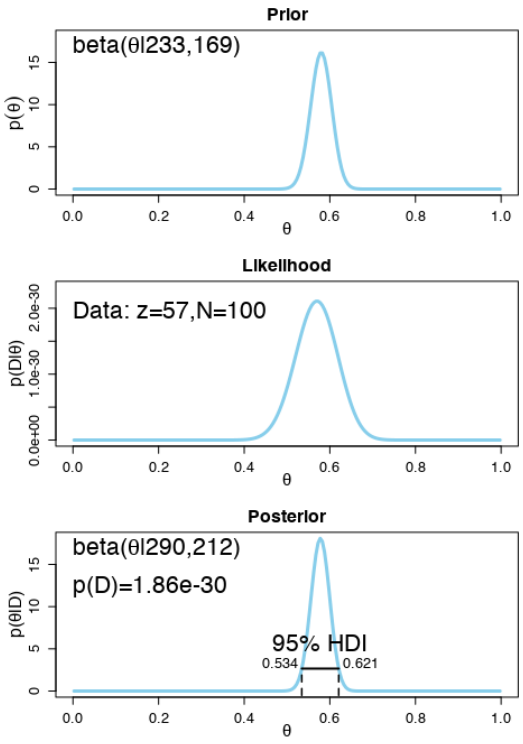
   Solution: the 95% HDI from the previous part shows that $\theta = 0.5$ is not among the credible values, hence it is not credible to believe that the population is equally divided in its preferences (at the 95%) level.

3. Say we conduct a follow-up poll to narrow our estimate of the population's preference. We randomly sample 100 people and find that 57 prefer candidate A. Assuming that the opinion of voters has not changed between polls, what is the 95% HDI on the posterior?

   Solution: using the previous posterior as a new prior, we obtain the following results.

```
post = BernBeta( post, c(rep(1,57), rep(0,43)))
```

```
[1] 290 212
```



   The 95% HDI for the preference still leans towards candidate A, but is a bit narrower, ranging from 0.534 to 0.621.

4. Based on the follow-up poll, is it credible to believe that the population is equally divided in its preferences among candidates?

   Solution: the 95% HDI from the previous results excludes $\theta = 0.5$; both the follow-up poll and the original poll suggest that the population is not equally divided (and actually prefers candidate A).

25.4.2 MCMC Methods

The true power of Bayesian inference is most keenly felt when the model specifications lead to a posteriors that cannot be manipulated analytically; in that case, it is usually possible to recreate a synthetic (or **simulated**) set of values that share the properties with a given posterior. Such processes are known as **Monte Carlo simulations**.

A **Markov chain** is an ordered, indexed set of random variables (a stochastic process) in which the values of the quantities at a given state depends probabilistically only on the values of the quantities at the preceding state.

Markov Chain Monte Carlo (MCMC) methods are a class of algorithms for sampling from a probability distribution based on the construction of a Markov chain with the desired distribution as its equilibrium distribution. The state of the chain after a number of steps is then used as a sample of the desired distribution.[24] MCMC techniques are often applied to solve integration and optimization problems in large-dimensional spaces.

24: The quality of the sample improves as a function of the number of steps.

These two types of problem play a fundamental role in machine learning, physics, statistics, econometrics and decision analysis. For instance, given variables $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ and data D, the following (typically intractable) integration problems are central to Bayesian inference:

- **normalization** – in order to obtain the posterior $P(\boldsymbol{\theta} \mid D; I)$ given the prior $P(\boldsymbol{\theta} \mid I)$ and likelihood $P(D \mid \boldsymbol{\theta}; I)$, the normalizing (denominator) factor in Bayes' theorem needs to be computed

$$P(\boldsymbol{\theta} \mid D; I) = \frac{P(\boldsymbol{\theta} \mid I) P(D \mid \boldsymbol{\theta}; I)}{\int_{\boldsymbol{\Theta}} P(D \mid \boldsymbol{\theta}; I) P(\boldsymbol{\theta} \mid I) \, d\boldsymbol{\theta}};$$

- **marginalization** – given the joint posterior of $(\boldsymbol{\theta}, \mathbf{x}) \in \boldsymbol{\Theta} \times \boldsymbol{\Omega}$, we may often be interested in the marginal posterior

$$P(\boldsymbol{\theta} \mid D; I) = \int P(\boldsymbol{\theta}, \mathbf{x} \mid D; I) \, d\mathbf{x};$$

- **expectation** – the final objective of the analysis is often to obtain summary statistics of the form

$$E[f(\boldsymbol{\theta})] = \int_{\boldsymbol{\Theta}} f(\boldsymbol{\theta}) P(\boldsymbol{\theta} \mid D; I) \, d\boldsymbol{\theta}$$

  for some function of interest (i.e., $f(\boldsymbol{\theta}) = \boldsymbol{\theta}$ or $f(\boldsymbol{\theta}) = (\boldsymbol{\theta} - E[\boldsymbol{\theta}])^2$, which represent the mean and the variance, respectively).

25.4.3 The MH Algorithm

The Metropolis-Hastings (MH) algorithm is a specific type of Monte Carlo process; it is likely among the ten algorithms that have had the greatest influence on the development and practice of science and engineering in recent years.[25]

25: The celebrated **Gibbs sampler** can be viewed as a special case of MH.

MH generates a random walk (that is, it generates a succession of posterior samples) in such a way that each step in the walk is **completely**

independent of the preceding steps; the decision to reject or accept the proposed step is also independent of the walk's history.

Any process for which the current step is independent (forgetful) of the previous states, namely

$$P(X_{n+1} = x \mid X_1 = x_1, \ldots, X_n = x_n; I) = P(X_{n+1} = x \mid X_n = x_n; I)$$

for all n, X_j and x_j, $j = 1, \ldots, n$, is called a **(first order) Markov process**, and a succession of such steps is a **(first order) Markov chain**.

MH uses a candidate or proposal distribution for the posterior, say $q(\cdot, \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a vector of parameters that is fixed by the user-called tuning parameters; MH then constructs a Markov Chain by proposing a value for $\boldsymbol{\theta}$ from this candidate distribution, and then either accepting or rejecting this value (with a certain probability).

Theoretically the proposal distributions can be nearly any distribution, but in practice it is recommended to keep to simple ones: a normal if the parameter of interest can be any real number (e.g., μ), or a log-normal if it has positive support (e.g., σ^2), say.

The MH algorithm **simulates** samples from a probability distribution by making use of the full joint density function and (independent) proposal distributions for each of the variables of interest.

Algorithm: Metropolis-Hastings Algorithm

Initialize $x^{(0)} \sim q(x)$
for $i = 1, 2, \cdots$ **do**
  Propose: $x^* \sim q(x^{(i)}|x^{(i-1)})$
  Acceptance Probability:

$$\alpha(x^*|x^{(i-1)}) = min\left\{1, \frac{q(x^{(i-1)}|x^*)\pi(x^*)}{q(x^*|x^{(i-1)})\pi(x^{(i-1)})}\right\}$$

  $u \sim U(0, 1)$
  if $u < \alpha$ **then**
    Accept the proposal: $x^{(i)} \leftarrow x^*$
  else
    Reject the proposal: $x^{(i)} \leftarrow x^{(i-1)}$
  end
end

The first step is to **initialize the sample value** for each random variable (often obtained by sampling from the variable's prior distribution). The main loop of the algorithm consists of three components:

1. **generate a candidate sample** x^* from the proposal distribution $q(x^{(i)}|x^{(i-1)})$;
2. **compute the acceptance probability** *via* the acceptance function $\alpha(x^*|x^{(i-1)})$ based on the proposal distribution and the full joint density $\pi(\cdot)$;
3. **accept the candidate sample** with probability α, the acceptance probability, or **reject it** otherwise.

Example (modified from [10, 6]): we use the MH algorithm to "learn" linear model parameters from a dataset. The **test data** for this example is generated as follows.

First, we establish the true model parameters.

```
set.seed(0)  # for replicability
t.A <- 10    # true slope
t.B <- 0     # true intercept
t.sd <- 20   # true noise
s.Size <- 50 # sample size
```

We will use equally spaced x values:

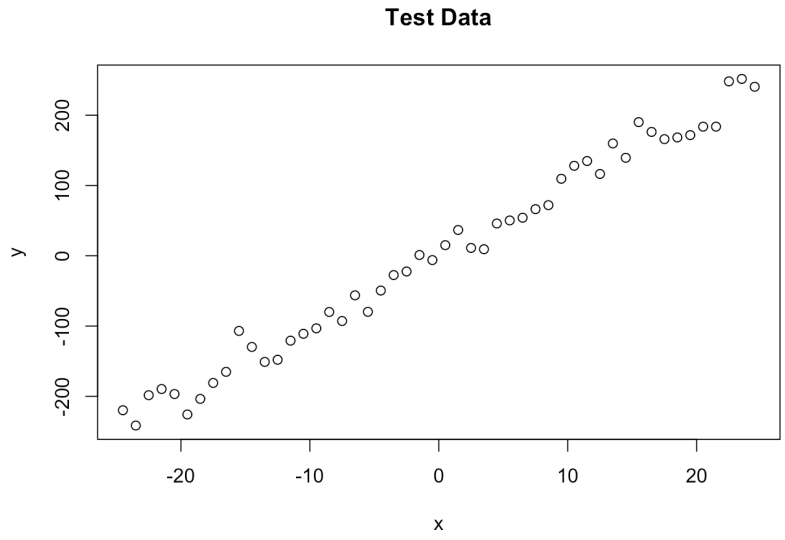
```
x <- (-(s.Size-1)/2):((s.Size-1)/2)
```

The corresponding y values are such that $y \sim \mathcal{N}(ax + b, \sigma^2)$:

```
y <-  t.A * x + t.B + rnorm(n=s.Size,mean=0,sd=t.sd)
```

The x values are balanced around zero in order to "de-correlate" the slope and the intercept.

```
plot(x,y, main="Test Data")
```



Defining the statistical model. The next step is to specify the statistical model. We already know that the data was created with a linear relationship $y = ax + b$ together with a normal error model $\mathcal{N}(0, \sigma^2)$, so we might as well use the same model for the fit and see if we can retrieve our original parameter values. Note however that, in general, the generating model is unknown.

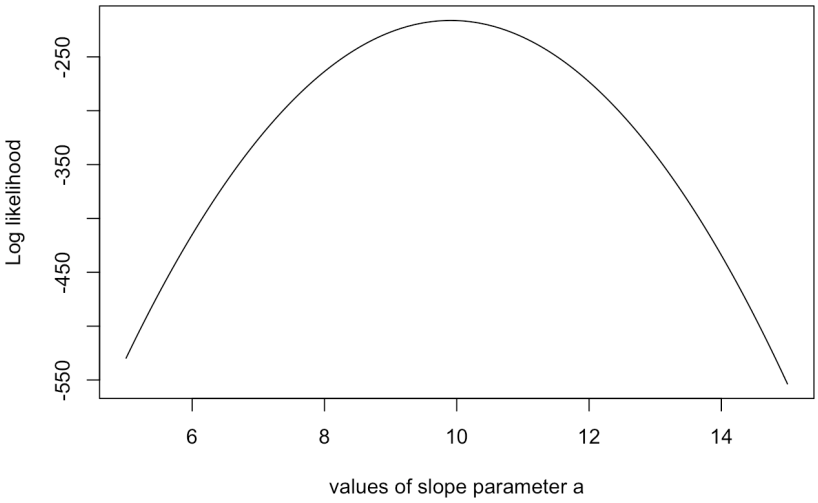
Deriving the likelihood function from the model. A linear model of the form $y = ax + b + \mathcal{N}(0, \sigma^2)$ takes the parameters (a, b, σ) as **inputs**. The **output** should be the probability of obtaining the test data under

this model: in this case, we only need to calculate the difference between the predictions $y = ax + b$ and the observed y, and then look up the probability (using `dnorm`) for such deviations to occur.

```
likehd <- function(param){
  a = param[1]
  b = param[2]
  sd = param[3]
  pred = a*x + b
  singlelikelihoods = dnorm(y, mean=pred, sd=sd, log=T)
  sumll = sum(singlelikelihoods)
  return(sumll)
}
```

For instance, we can find and plot the likelihood profile of the slope:

```
s.values <- function(x){return(likehd(c(x, t.B, t.sd)))}
s.likehds <- lapply(seq(1/2*t.A, 3/2*t.A, by=.05), s.values )
plot (seq(1/2*t.A, 3/2*t.A, by=.05), s.likehds , type="l",
  xlab = "values of slope parameter a", ylab = "Log likelihood")
```



Defining the priors. In Bayesian analysis, the next step is always required: we have to specify a prior distribution for each of the model parameters. To keep things simple, we will use a uniform distribution for the slope, and normal distributions for the noise and the intercept.[26]

26: We will work with the logarithms of all quantities, so that the likelihood is a sum and not a product as would usually be the case.

```
prior <- function(param){
  a = param[1]
  b = param[2]
  sd = param[3]
  aprior = dunif(a, min=0, max=2*t.A, log = T)
  bprior = dnorm(b, mean=t.B, sd = 5, log = T)
  sdprior = dunif(sd, min=0, max=2*t.sd, log = T)
  return(aprior+bprior+sdprior)
}
```

The posterior. The product of prior by likelihood is the actual quantity that MCMC works with (it is not, strictly speaking, the posterior as it is not normalized).

```
posterior <- function(param){
  return (likehd(param) + prior(param))
}
```

Applying the MH algorithm. One of the most frequent applications of MH (as in this example) is sampling from the posterior density in Bayesian statistics.[27]

27: The algorithm may be used to sample from any integrable function.

The aim of the algorithm is to jump around in parameter space, but in such a way as to have the probability to land at a point be proportional to the function we sample from (this is usually called the **target function**). In this case, the target function is the posterior that was defined previously.

This is achieved by

1. starting with a random parameter vector;
2. choosing a new parameter vector near the old value based on some probability density (the proposal function), and
3. jumping to this new point with a probability

$$\alpha = \min\{1, g(\text{new})/g(\text{old})\},$$

where g is the target.

The distribution of the parameter vectors MH visits converges to the target distribution g.

```
proposalfunction <- function(param){
  return(rnorm(3,mean = param, sd= c(0.1,0.5,0.3)))
}

run_metropolis_MCMC <- function(startvalue, iterations){
  chain = array(dim = c(iterations+1,3))
  chain[1,] = startvalue
  for (i in 1:iterations){
    proposal = proposalfunction(chain[i,])

    probab = exp(posterior(proposal) - posterior(chain[i,]))
    if (runif(1) < probab){
      chain[i+1,] = proposal
    }
    else{
      chain[i+1,] = chain[i,]
    }
  }
  return(chain)
}

startvalue = c(4,1,10) # random choice
chain = run_metropolis_MCMC(startvalue, 10000)
```

The first steps of the algorithm may be biased by the initialization process; they are usually discarded for the analysis (this is referred to as the **burn-in time**).

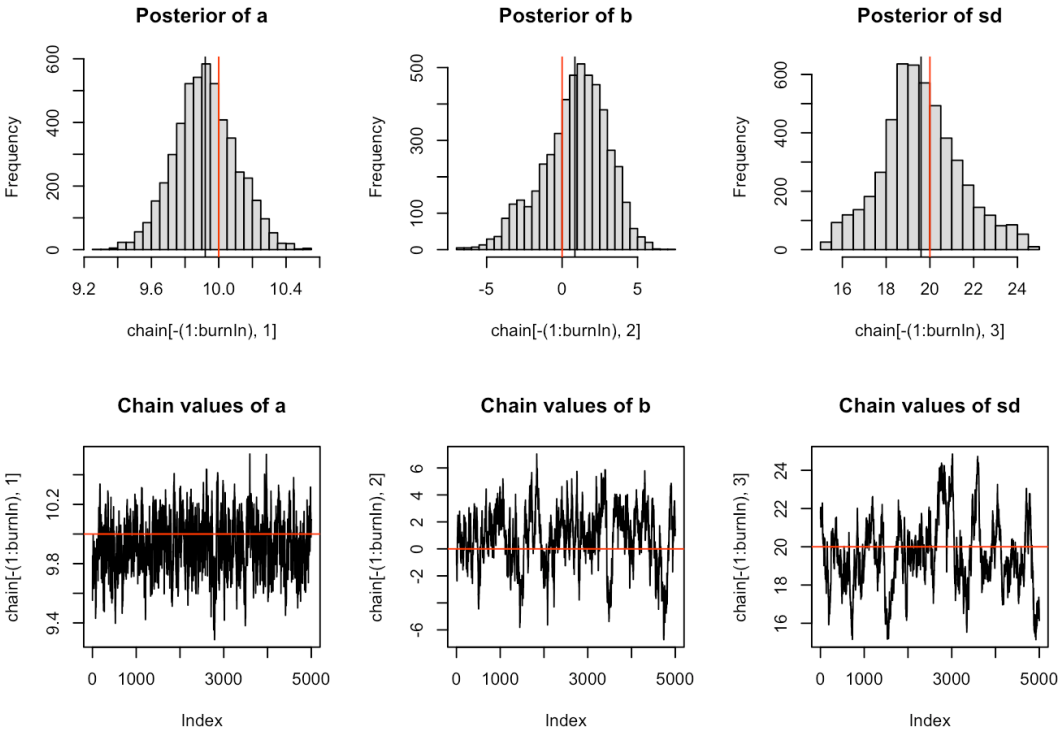
```
burnIn = 5000
acceptance = 1-mean(duplicated(chain[-(1:burnIn),]))
```

The **acceptance rate** is an interesting output to study: how often was a proposal rejected by the MH acceptance criterion? The acceptance rate can be influenced by the proposal function: generally, the nearer the proposal is to the latest value, the larger the acceptance rate.[28]

We plot the results below (the true parameter values are shown in red).

28: Very high acceptance rates, however, are usually not beneficial, as this implies that the algorithms is "staying" in the same neighbourhood, which results in s**ub-optimal probing of the parameter space** (there is very little **mixing**). Acceptance rates between 20% and 30% are considered optimal for typical applications [7].

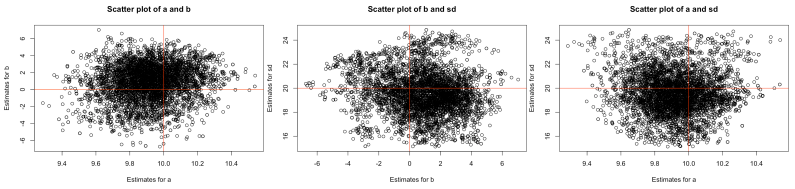
```
par(mfrow = c(2,3))
hist(chain[-(1:burnIn),1],nclass=30, main="Posterior of a")
abline(v = mean(chain[-(1:burnIn),1]))
abline(v = t.A, col="red" )
hist(chain[-(1:burnIn),2],nclass=30, main="Posterior of b")
abline(v = mean(chain[-(1:burnIn),2]))
abline(v = t.B, col="red" )
hist(chain[-(1:burnIn),3],nclass=30, main="Posterior of sd")
abline(v = mean(chain[-(1:burnIn),3]) )
abline(v = t.sd, col="red" )
plot(chain[-(1:burnIn),1], type = "l", main = "Chain values of a")
abline(h = t.A, col="red" )
plot(chain[-(1:burnIn),2], type = "l", main = "Chain values of b")
abline(h = t.B, col="red" )
plot(chain[-(1:burnIn),3], type = "l", main = "Chain values of sd")
abline(h = t.sd, col="red" )
```



The upper row shows posterior estimates for the slope a, intercept b, and standard deviation of the error σ; the lower row shows the Markov Chain of parameter values. We retrieve (more or less) the original parameters that were used to create the data, and there is a certain area around the highest posterior values that also show some support by the data, which is the Bayesian equivalent of confidence intervals.

These posterior distributions are **marginal distributions**; the pairwise joint distributions are shown below (again, with true parameter values in red – the horizontal and vertical lines).

```
plot(chain[-(1:burnIn),1:2], main="Scatter plot of a and b",
     xlab="Estimates for a", ylab="Estimates for b")
abline(v = t.A, col="red" )
abline(h = t.B, col="red" )
plot(chain[-(1:burnIn),2:3], main="Scatter plot of b and sd",
     xlab="Estimates for b", ylab="Estimates for sd")
abline(v = t.B, col="red" )
abline(h = t.sd, col="red" )
plot(chain[-(1:burnIn),c(1,3)], main="Scatter plot of a and sd",
     xlab="Estimates for a", ylab="Estimates for sd")
abline(v = t.A, col="red" )
abline(h = t.sd, col="red" )
```



The posterior distributions certainly do seem to contain the true parameter values. By way of comparison, a simple linear regression analysis would yield the following estimates:

```
summary(lm(y~x))
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-33.067 -12.201  -3.733  14.562  46.192

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.4786     2.6115   0.183    0.855
x             9.9082     0.1810  54.751   <2e-16 ***
---
Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1

Residual standard error: 18.47 on 48 degrees of freedom
Multiple R-squared:  0.9842,    Adjusted R-squared:  0.9839
F-statistic:  2998 on 1 and 48 DF,  p-value: < 2.2e-16
```

Which method is best, in this context? What are the advantages and disadvantages of each?

25.5 Additional Topics

According to [5],

> the central feature of Bayesian inference is the direct quantification of uncertainty.

Bayesian approach to modeling uncertainty is particularly useful when:

- the available data is **limited**;
- there is some concern about **overfitting**;
- some facts are **more likely to be true** than others, but that information is not contained in the data, or
- the precise likelihood of certain facts is more important than solely determining which fact is most likely (or least likely).

As discussed previously, Bayesian methods have a number of powerful features. They allow analysts to:

- incorporate **specific knowledge** about parameters of interest;
- logically **update knowledge** about the parameter after observing sample data;
- make **formal probability statements** about parameters of interest;
- specify **model assumptions** and check model quality and sensitivity to these assumptions in a straightforward manner, and
- provide **probability distributions** rather than point estimates.

25.5.1 Uncertainty

The following example represents a Bayesian approach to dealing with the uncertainty of the so-called **envelope paradox**.

Example: you are given two indistinguishable envelopes, each containing a cheque, one being twice as much as the other. You may pick one envelope and keep the money it contains. Having chosen an envelope at will, but before inspecting it, you are given the chance to switch envelopes. Should you switch? What is the expected outcome in doing so? Explain how this game leads to infinite cycling.

Solution: let V be the (unknown) value found in the envelope after the first selection. The other envelope then contains either $\frac{1}{2}V$ or $2V$, both with probability 0.5, and the expected value of trading is

$$E[\text{trade}] = 0.5 \times \frac{1}{2}V + 0.5 \times 2V = \frac{5}{4}V > V;$$

and so it appears that trading is advantageous.

Let the (still unknown) value of the cheque in the new envelope be W. The same argument shows that the expected value of trading *that* envelope is $\frac{5}{4}W > W$, so it would make sense to trade the envelope once more, and yet once more, and so on, leading to infinite cycling.

There is a Bayesian approach to the problem, however. Let V be the (uncertain) value in the original selection, and W be the (also uncertain) value in the second envelope. A proper resolution requires a joint (prior) distribution for V and W. Now, in the absence of any other information,

the most we can say about this distribution using the maximum entropy principle is that $P(V < W) = P(V > W) = 0.5$.

By definition, if $V < W$, then $W = 2V$; if, on the other hand, $V > W$ then $W = \frac{V}{2}$. We now show that the expected value in both envelopes is the same, and thus that trading envelope is no better strategy than keeping the original selection. Using Bayes' Theorem, we compute that

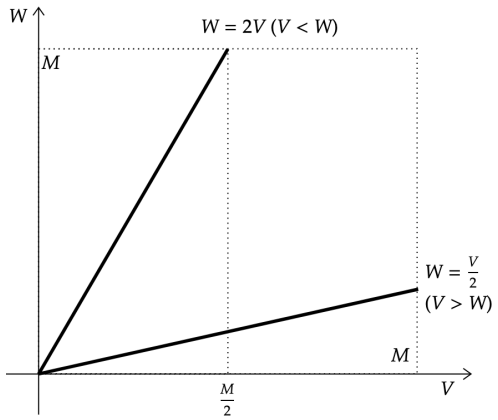
$$\begin{aligned} E[W] &= E[W|V < W]P(V < W) + E[W|V > W]P(V > W) \\ &= E[2V|V < W] \cdot 0.5 + E[0.5V|V > W] \cdot 0.5 \\ &= E[V|V < W] + 0.25 \cdot E[V|V > W], \end{aligned}$$

while

$$\begin{aligned} E[V] &= E[V|V < W]P(V < W) + E[V|V > W]P(V > W) \\ &= 0.5 \cdot E[V|V < W] + 0.5 \cdot E[V|V > W]. \end{aligned}$$

Before we can proceed any further, we must have some information about the joint distribution $P(V, W)$ (note, however, that $E[W]$ will not typically be equal to $\frac{5}{4}V$, as had been assumed at the start of the solution). The domain Ω of the joint probability consists of those pairs (V, W) satisfying $V = 2W$ $(V > W)$ or $W = 2V$ $(V < W)$ for $0 < V, W < M$, where $M < \infty$ is some upper limit on the value of each cheque.[29]

29: In the worst case scenario, M would have to be smaller than the total amount of wealth available to humanity throughout history, although in practice M should be substantially smaller. Obviously, a different argument will need to be made in the case $M = \infty$.



We have assumed that the probability weight on each branch of Ω is $1/2$; if we further assume, say, that the cheque value is as likely to be any of the allowable values on these branches, then the joint distribution is

$$P(V, W) = \begin{cases} \frac{1}{M} & \text{if } V < W \\ \frac{1}{2M} & \text{if } V > W \\ 0 & \text{otherwise} \end{cases}$$

and the expectations listed above are

$$E[V|V < W] = \int_{V<W} V \cdot P(V, W)\, d\Omega = \int_0^{M/2} V \cdot \frac{1}{M}\, dV = \frac{M}{8}$$

and

$$E[V|V > W] = \int_{V>W} V \cdot P(V, W)\, d\Omega = \int_0^{M} V \cdot \frac{1}{2M}\, dV = \frac{M}{4}.$$

Therefore,

$$E[W] = \frac{M}{8} + 0.25 \cdot \frac{M}{4} = \frac{3M}{16}$$

and

$$E[V] = 0.5 \cdot \frac{M}{8} + 0.5 \cdot \frac{M}{4} = \frac{3M}{16},$$

and switching the envelope does not change the expected value of the outcome. There is no paradox; no infinite cycling.

Example: After the sudden death of her two baby sons, Sally Clark was sentenced by a U.K. court to life in prison in 1996. Among other errors, expert witness Sir Roy Meadow had wrongly interpreted the small probability of two cot deaths as a small probability of Clark's innocence. After a long campaign, which included the refutation of Meadow's statistics using Bayesian statistics, Clark was released in 2003. While Clark's innocence could not be proven beyond the shadow of a doubt using such methods, her culpability could also not be established beyond reasonable doubt and she was cleared.[30]

30: An informative write-up of the situation can be found online [2].

25.5.2 Bayesian A/B Testing

A/B **testing** is an excellent tool for deciding whether or not to roll out incremental features. To perform an A/B test, we divide users randomly into a **test** group and into a **control** group, then provide the new feature to the test group while letting the control group continue to experience the current version of the product.

If the randomization procedure is appropriate, we may be able to attribute any difference in outcomes between the two groups to the changes we are rolling out without having to account for other sources of variation affecting the user behaviour. Before acting on these results, however, it is important to understand the likelihood that any observed differences is merely due to chance rather than to product modification.

For example, it is perfectly possible to obtain different H/T ratios between two fair coins if we only conduct a limited number of tosses; In the same manner, it is possible to observe a change between the A and B groups even if the underlying user behavior is identical.

Example: (modified from [13]) *Wakefield Tiles* is a company that sells floor tiles by mail order. They are trying to become an active player into the lucrative Chelsea market by offering a new type of tile to the region's contractors.

The marketing department have conducted a pilot study and tried two different marketing methods:

- A – sending a colourful brochure in the mail to invite contractors to visit the company's showroom;
- B – sending a colourful brochure in the mail to invite contractors to visit the company's showroom, while including free tile samples.

The marketing department sent out 16 mail packages of type A and 16 mail packages of type B. Four Chelseaites that received a package of type A visited the showroom, while 8 of those receiving a package of type B did the same.

The company is aware that:

- a mailing of type A costs 30$ (printing cost and postage);
- a mailing of type B costs 300$ (also includes the cost of the free tile samples);
- a visit to the showroom yields, on average, 1000$ in revenue during the next year.

Which of the methods (A or B) is most advantageous to *Wakefield Tiles*?

Solution: the Bayesian solution requires the construction of a prior distribution and of a **generative model**; as part of the generative model, we will need to produce n replicates of samples from the binomial distribution.[31]

31: Which can be achieved in `R` using `rbinom(n,size,prob)`.

The binomial distribution simulates `n` times the number of "successes" when performing `size` trials (mailings), where the probability of a "success" is `prob`. A commonly used prior for `prob` is the uniform distribution $U(0,1)$, from which we sample in `R` *via* `runif(1, min = 0, max = 1)`.

We start by setting a seed for replicability, and set the number of replicates (trials).

```
set.seed(1111) # for replicability
n.draws <- 200000
```

Next, we generate a probability of success for mailings A and B, for each of the replicates.

```
prior <- data.frame(p.A = runif(n.draws, 0, 1),
                    p.B = runif(n.draws, 0, 1))
```

The generative model tells us how many visitors to expect for mailing types A, B, for each replicate.

```
generative.model <- function(p.A, p.B) {
  visitors.A <- rbinom(1, 16, p.A)
  visitors.B <- rbinom(1, 16, p.B)
  c(visitors.A = visitors.A, visitors.B = visitors.B)
}
```

We then simulate data using the parameters from the prior and the generative model. This yields the actual number of visitors for each replicate.

```
sim.data <- as.data.frame( t(sapply(1:n.draws, function(i) {
  generative.model(prior$p.A[i], prior$p.B[i])})))
```

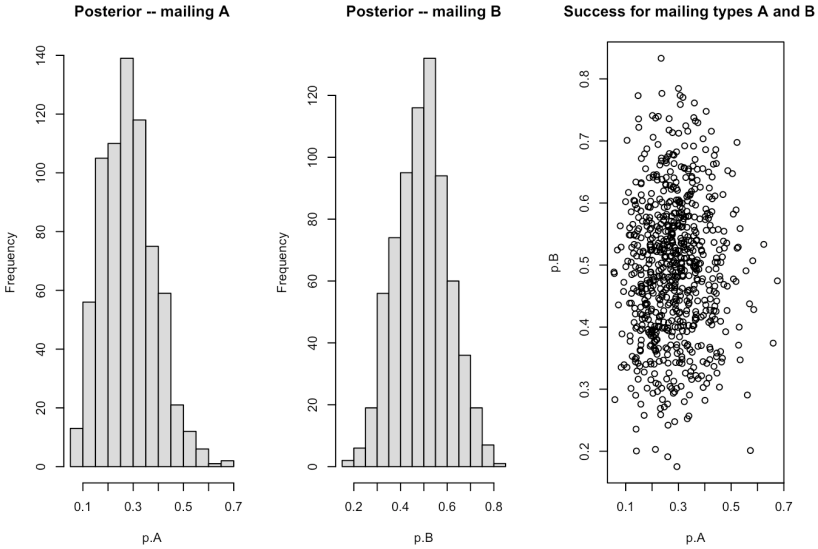
Only those prior probabilities for which the generative model match the observed data are retained.

```
posterior <- prior[sim.data$visitors.A == 4 &
                   sim.data$visitors.B == 8, ]
```

In this case, there are enough trials to ensure that the posterior is non-empty; what could be done if that was not the case?

Finally, we visualize the posteriors:

```
par(mfrow = c(1,3))
hist(posterior$p.A, main = "Posterior -- mailing A",
     xlab="p.A")
hist(posterior$p.B, main = "Posterior -- mailing B",
     xlab="p.B")
plot(posterior,main = "Success for mailing types A and B",
     xlab="p.A", ylab="p.B")
```

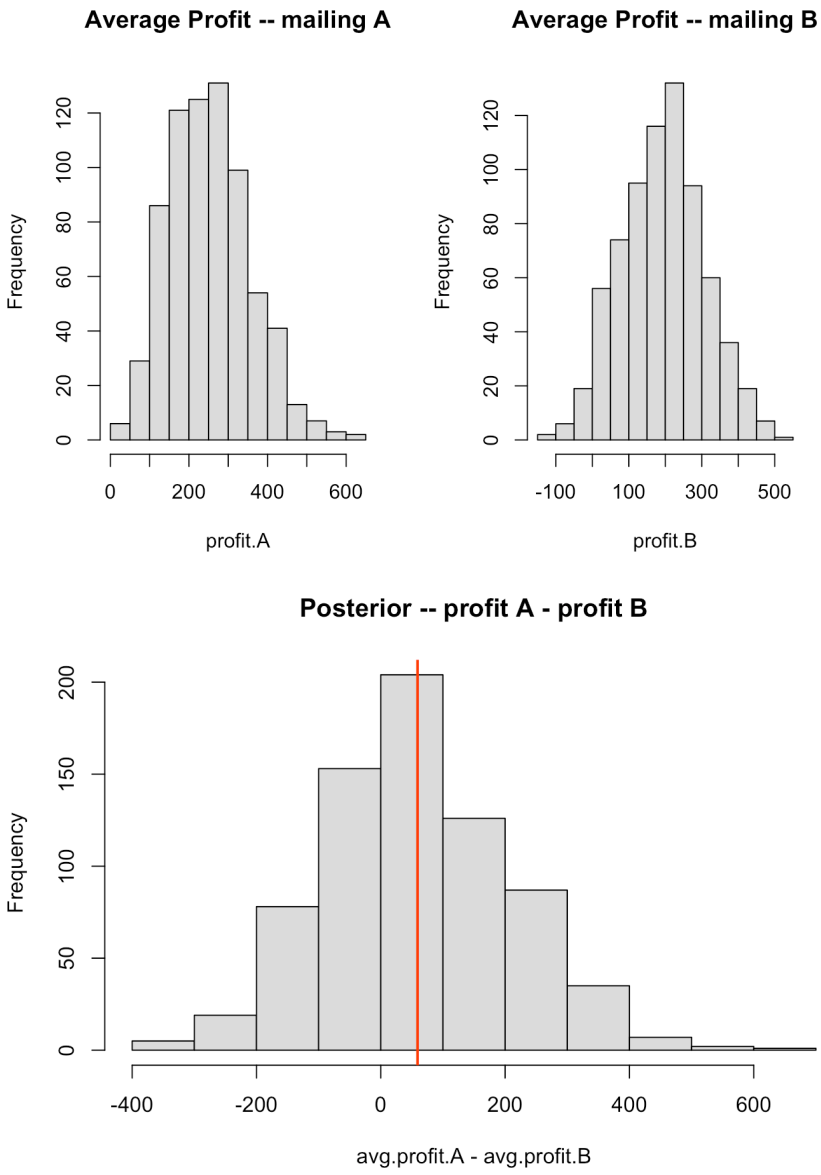


The posterior distributions for the probability of success for each mailing types are produced as below (see next page for display).

```
par(mfrow = c(1,2))
avg.profit.A <- -30 + posterior$p.A * 1000
avg.profit.B <- -300 + posterior$p.B * 1000
hist(avg.profit.A, main = "Average Profit -- mailing A",
     xlab="profit.A")
hist(avg.profit.B, main = "Average Profit -- mailing B",
     xlab="profit.B")
```

In order to estimate the average profit for each mailing type, we use the posterior distributions for the probability of success (see next page).

```
hist(avg.profit.A - avg.profit.B, main="Posterior --
     profit A - profit B")
(expected.avg.profit.diff <- mean(avg.profit.A - avg.profit.B))
abline(v = expected.avg.profit.diff , col = "red", lwd =2)
```




```
[1] 59.13869
```

The expected profit for mailing type A is about 60$ higher than for mailing type B (numbers may vary, depending on the seed). Keeping it simple seems to be a better idea in this context.

25.6 Exercises

1. In many situations, researchers observe that a certain region of the brain is active and infer that a particular cognitive function is therefore being carried out; [12] cautioned that such inferences are not necessarily firm and need to be made with Bayes' rule in mind. The same paper reports the following frequency table of previous studies that involved any language-related task (specifically phonological and semantic processing) and whether or not a particular **region of interest** (ROI) in the brain was activated (see table below). Suppose that a new study is conducted and finds that the ROI is activated (A). If the prior probability that the task involves language processing is $P(L) = 0.5$, what is the posterior probability, $P(L \mid A)$, given that the ROI is activated?

	Language (L)	Other ($\overline{L}$)
Activated (A)	166	199
Not Activated ($\overline{A}$)	703	2154

2. Suppose that, in 1975, 52% of UK voters supported the Labour Party and 48% the Conservative Party. Suppose further that 55% of Labour voters wanted the UK to remain part of the EEC and 85% of Conservative voters were also in favour. What is the probability that a person voting "Yes" (in favour of remaining in the EEC) in the 1975 referendum is a Labour voter? [8]
3. Given the following statistics, what is the probability that a woman over 50 years of age has breast cancer if she receives a positive mammogram result? [Bayes' Theorem Problems, Definition and Examples]

    a) 1% of women over 50 have breast cancer;
    b) 90% of women over 50 who have breast cancer test positive on mammograms;
    c) 8% of women over 50 will obtain a false positive result on a breast cancer test.

4. What would it take for you to update ...

    a) your belief in the existence/non-existence of a deity?
    b) your belief in the shape of the Earth?
    c) your political affiliation?
    d) your allegiance to a sport team? (Go Sens!)
    e) your belief in the effectiveness of homeopathic remedies?
    f) your belief in the effectiveness of Bayesian analysis?

5. Suppose that a test for a particular disease has a very high success rate. When a patient has the disease, the test accurately reports a 'positive' with probability 0.99; when they do not, the test accurately reports a 'negative' with probability 0.95. Assume further that only 0.1% of the population has the disease. What is the probability that a patient who tests positive does not in fact have the disease?
6. A road safety analyst has access to a dataset of fatal vehicle collisions (such as Canada's *National Collision Database*) on roads in a specific region. The dataset is built using police reports, and it contains relevant collision information such as: the severity of the collision, the age of the drivers, the number of passengers in each vehicle, the date and time of the collision, weather and road conditions, blood alcohol content (BAC), etc. Let us further assume that the analyst has access to aggregated weather data and R.I.D.E. (sobriety checkpoint) reports for that region. Some information may be missing from the police reports at a given moment (perhaps the coroner has not yet had the chance to determine the BAC level, or some of the data may have been mistakenly erased and/or corrupted). For some collisions, we may need to answer either or both of the following questions: did alcohol play a role in the collision? did "bad" weather play a role in the collision? As usual, let I denote all relevant information relating to the situation, such as the snowy months of the year, the incidence of impaired driving in that region, etc. The analyst will consider 3 propositions:

    a) A: a fatal collision has occurred
    b) B: the weather and road conditions were bad
    c) C: the BAC level of one of the drivers involved in a collision was above 0.08% per volume

The analysts may have an interest in $P(B \mid A; I)$, $P(C \mid A; I)$, $P(B, C \mid A; I)$, $P(B, -C \mid A; I)$, or $P(-B, C \mid A; I)$. Derive an expression to compute the probability that "bad" weather and road conditions were present at the time of the collision.

7. A **Mild Winter** scenario (we use the set-up of question 6): during a mild winter, "bad" weather affected regional road conditions 5% of the time. The analyst knows from other sources that the probabilities of fatal collisions given "bad" and "good" weather conditions in the region over the winter are 0.01% and 0.002%, respectively. If a fatal collision occurred on a regional road that winter, what is the probability that the weather conditions were "bad" on that road at that time? Is the result surprising?
8. **Not Quite as Mild a Winter** scenario (we use the set-up of questions 6 and 7): assume that the winter was not quite as mild (perhaps "bad" weather affected regional road conditions 10% of the time, say). If a fatal collision occurred on a regional road that winter, what is the probability that the weather conditions were "bad" on that road at that time? How much of a jump are you expecting compared to question 7?
9. Use the set-up of questions 6-8. Just how rough of a winter would be necessary before we conclude that a given fatal collision was more likely to have occurred in "bad" weather?
10. Use the set-up of questions 6-9. In what follows, we assume that the analyst does not have access to other sources from which to derive the individual probabilities of fatal collisions given "bad" and "good" weather conditions in the region. Instead, the analyst has access to data that suggests that the probability of a fatal collision in "bad" weather is k times as high as the probability of a fatal collision in "good" weather. Let the probability of "bad" weather be $w \in (0, 1)$. Derive an expression for the probability that the weather conditions were "bad" on that road at that time, given that a fatal collision occurred, in terms of k and w.
11. **Really Rough Winter** scenario (see questions 6-10): during a really rough winter, "bad" weather affected road conditions with probability $w = 0.2$. Determine the probabilities that there were "bad" weather conditions given a fatal collision under 4 different values: $k = 0.1, 1, 10, 100$. Which of these scenarios is most likely?
12. Use the set-up of questions 6-11. In the next scenario, we assume that the traffic flow changes depending on the weather; while some individuals need to be on the roads no matter the conditions, others might tend to avoid the roads when the conditions are "bad". Make whatever assumptions are necessary and analyze the situation as you have done in the previous questions.
13. Use the set-up of questions 6-12. Repeat the process for the other conditional probabilities of interest.
14. A lifetime's supply of poutine is placed randomly behind one of three identical doors. The other two doors lead to empty rooms. You are asked to pick a door. One of the doors you have not selected is opened, revealing an empty room. You are given the option of changing your pick. What is your optimal strategy?

    a) Determine the ideal strategy using a simulation.
    b) Analyze a similar situation (for 100 doors instead of 3) using Bayes' Theorem.
    c) Analyze the situation using Bayes' Theorem.

15. How many heads in a row would you need to observe before you would start doubting whether a coin is fair or not?
16. Estimate the parameters (μ_i, σ_i) for $i = 1, \ldots, 12$ in the Salary example.
17. Play with the parameters and implement new scenarios for the Money (Dollar Bill Y'All) example.
18. Play with the `BernBeta()` function. Do you spot anything surprising?
19. Suppose you have in your possession a coin that you know was minted by the federal government and for which you have no reason to suspect tampering of any kind. Your prior belief about fairness of the coin is thus strong. You flip the coin 10 times and record 9 H(eads). What is your predicted probability of obtaining 1H on the 11th flip? Explain your answer carefully; justify your choice of prior. How would your answer change (if at all) if you use a frequentist viewpoint?
20. A mysterious stranger hands you a different coin, this one made of some strange-to-the-touch material, on which the words "Global Tricksters Association" You flip the coin 10 times and once again record 9H. What is your predicted probability of obtaining 1H on the 11th flip? Explain your answer carefully; justify your choice of prior. Hint: what would be a reasonable prior for this scenario?

21. A group of adults are doing a simple learning experiment: when they see the two words "radio" and "ocean" appear simultaneously on a computer screen, they are asked to press the F key on the keyboard; whenever the words "radio" and "mountain" appear on the screen, they are asked to press the J key. After several practice repetitions, two new tasks are introduced: in the first, the word "radio" appears by itself and the participants are asked to provide the best response (F or J) based on what they learned before; in the second, the words "ocean" and "mountain" appear simultaneously and the participants are once again asked to provide the best response. This is repeated with 50 people. The data shows that, for the first test, 40 participants answered with F and 10 with J; while for the second test, 15 responded with F and 35 with J. Are people biased toward F or toward J for either of the two tests? To answer this question, assume a uniform prior, and use a 95% HDI to decide which biases can be declared to be credible.
22. Suppose that the marketing group of a company is testing a new web page, with the hope of increasing the conversion rate (proportion of visitors who sign up or take some other action). The data is collected in the file `ab_data.csv`, which lists user visits with whether they were sent to the new page or the old page, and whether there was a conversion.
    a) Explore and visualize the dataset.
    b) We conduct Bayesian A/B testing, by defining and updating independent priors on the old and new conversion rates, to arrive at respective posterior distributions for the old page and the new page. Try a prior of `Beta(alpha=2, beta=20)` for the old rate, which represents what has been observed in the past. Start with a subset of 100 data points and perform inference. Find the posterior probability that the new page has a higher conversion rate. Hint: use random samples from the independent posteriors to estimate the probability. Update the posteriors with another 100 data points. At what data size do the priors become irrelevant?
23. Sometimes we don't just want to estimate a dependent variable, we want a probability distribution for it. For instance, if one's life expectancy is 80 years, we might want to know whether it's a 50/50 split between 0 years and 160 years, or some other distribution.
    a) Load the `mimic3d.csv` dataset which lists the length of stay in a hospital (`LOSdays`) along with a number of other variables. Explore and visualize this dataset.
    b) Construct a dataset `patients.csv` containing information about 10 or so (or more) "patients", for all but the `LOSdays` variable (you may use friends and family members, classmates, etc. as a basis for your observations).
    c) Predict the length of the hospital stay for the patients in the dataset by conducting a Bayesian linear regression analysis. What's the probability of staying longer than 2 days and therefore definitely missing work? Use normal priors for simplicity.

Chapter References

[1] T. Bayes. 'An essay towards solving a problem in the doctrine of chances'. In: *Phil. Trans. of the Royal Soc. of London* 53 (1763), pp. 370–418.

[2] Bayesians Without Borders. *Sally Clark is Wrongly Convicted of Murdering Her Children* .

[3] R.T. Cox. 'Probability, Frequency, and Reasonable Expectation'. In: *American Journal of Physics* 14.1 (1946).

[4] S. DeDeo. *Maximum Entropy Methods Tutorial. A Simple Example: The Taxicab (video)* .

[5] A. Gelman et al. *Bayesian Data Analysis (3rd ed.)* CRC Press, 2013.

[6] F. Hartig. 'A simple Metropolis-Hastings MCMC in R '. In: ().

[7] F. Hartig. 'MCMC chain analysis and convergence diagnostics with coda in R '. In: ().

[8] D. Hitchcock. *Introduction to Bayesian Data Analysis (course notes)*. Department of Statistics, University of South Carolina, 2014.

[9] E.T. Jaynes. *Probability Theory: the Logic of Science*. Cambridge Press, 2003.

[10] J.K. Kruschke. *Doing Bayesian Data Analysis: a Tutorial with R, JAGS, and Stan (2nd ed.)* Academic Press, 2011.

[11] T. Oliphant. *A Bayesian perspective on estimating mean, variance, and standard-deviation from data* . All Faculty Publications, BYU. 2006.

[12] R.A. Poldrac. 'Can cognitive processes be inferred from neuroimaging data?' In: *Trends Cogn. Sci.* (2006).

[13] rasmusab. *Bayesian A/B Testing for Swedish Fish Incorporated* . Tutorial.

[14] N. Silver. *The Signal and the Noise*. Penguin, 2012.

[15] T. Stoppard. *Rosencrantz and Guildenstern Are Dead*. Grove Atlantic, 1966.

[16] Wikipedia. *Conjugate Priors* .

Anomaly Detection and Outlier Analysis

26

by **Patrick Boily**, with contributions from **Youssouph Cissokho**, **Soufiane Fadel**, and **Richard Millson**

With the advent of automatic data collection, it is now possible to store and process large troves of data. There are technical issues associated to massive data sets, such as the speed and efficiency of analytical methods, but there are also problems related to the detection of anomalous observations and the analysis of outliers.

Extreme and irregular values behave very differently from the majority of observations. For instance, they can represent criminal attacks, fraud attempts, targeted attacks, or data collection errors. As a result, anomaly detection and outlier analysis play a crucial role in cybersecurity, quality control, etc. [1, 24, 11]. The (potentially) heavy human price and technical consequences related to the presence of such observations go a long way towards explaining why the topic has attracted attention recently.

In this chapter, we review various detection methods, with particular attention paid to both supervised and unsupervised methods.

26.1 Overview[1]

1: This section is an extension of Section 15.5.

Isaac Asimov, the prolific American author, once wrote that

> The most exciting phrase to hear [...], the one that heralds the most discoveries, is not "Eureka!" but "That's funny ... ".

However, anomalous observations are not only harbingers of great scientific discoveries – unexpected observations can spoil analyses or be indicative of the presence of issues related to data collection or data processing.[2]

2: Throughout, the definitions of terms like **normal** and **anomalous** will be kept purposely vague, to allow for increased flexibility.

Either way, it becomes imperative for decision-makers and analysts to establish anomaly detection protocols, and to identify strategies to deal with such observations.

26.1.1 Basic Notions and Concepts

Outlying observations are data points which are **atypical** in comparison to the unit's remaining features (*within-unit*), or in comparison to the measurements for other units (*between-units*), or as part of a collective subset of observations. Outliers are thus observations which are **dissimilar to other cases** or which contradict **known dependencies** or rules.[3]

3: Outlying observations may be anomalous along any of the individual variables, or in combinations of variables.

Observations could be anomalous in one context, but not in another.

Consider, for instance, an adult male who is 6-side tall. Such a man would fall in the 86th percentile among Canadian males [16], which, while on the tall side, is not unusual; in Bolivia, however, the same man would land in the 99.9th percentile, which would mark him as extremely tall and quite dissimilar to the rest of the population.

Anomaly detection points towards **interesting questions** for analysts and subject matter experts: in this case, why is there such a large discrepancy in the two populations?

In practice, an **outlier/anomalous observation** may arise as

- a **"bad" object/measurement:** data artifacts, spelling mistakes, poorly imputed values, etc.
- a **misclassified observation:** according to the existing data patterns, the observation should have been labeled differently;
- an observation whose measurements are found in the **distribution tails**, of a large enough number of features;
- an **unknown unknowns**: a completely new type of observations whose existence was heretofore unsuspected.

A common mistake that analysts make when dealing with outlying observations is to remove them from the dataset without carefully studying whether they are **influential data points**, that is, observations whose absence leads to **markedly different** analysis results.

When influential observations are identified, **remedial measures**[4] may need to be applied to minimize any undue effect. Note that outliers may be influential, and influential data points may be outliers, but the conditions are neither necessary nor sufficient.

4: Such as data transformation strategies.

Anomaly Detection By definition, anomalies are **infrequent**,[5] which makes it difficult to distinguish them from banal **noise** or **data collection errors**.

5: And shrouded in **uncertainty** due to their relatively low numbers.

Furthermore, the boundary between normal and deviant observations is usually **fuzzy**; with the advent of e-shops, for instance, a purchase which is recorded at 3AM local time does not necessarily raise a red flag anymore.

When anomalies are actually associated to **malicious activities**, they are more than often **disguised** in order to blend in with normal observations, which obviously complicates the detection process. Numerous methods exist to identify anomalous observations; **none of them are foolproof** and judgement must be used.

Methods that employ graphical aids (such as box-plots, scatterplots, scatterplot matrices, and 2D tours) to identify outliers are particularly easy to implement, but a low-dimensional setting is usually required for ease of interpretability. These methods usually find the anomalies that **shout the loudest** [6].

Analytical methods also exist (using Cooke's or Mahalanobis' distances, say), but in general some additional level of analysis must be performed, especially when trying to identify influential observations (*cf.* **leverage**).

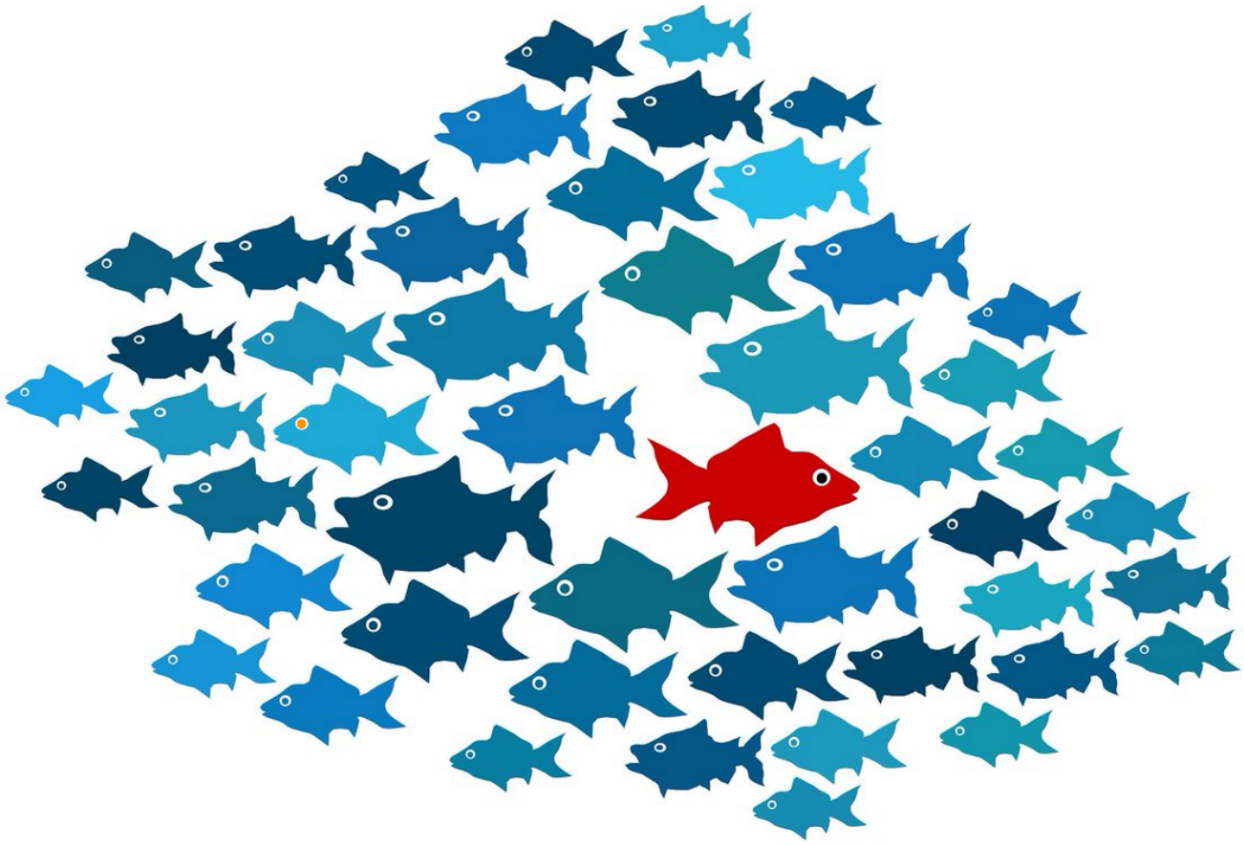


Figure 26.1: A school of fish: what jumps at you here? [author unknown]

With small datasets, anomaly detection can be conducted on a case-by-case basis, but with large datasets, the temptation to use **automated detection/removal** is strong – care must be exercised before the analyst decides to go down that route. This stems partly from the fact that once the "anomalous" observations have been removed from the dataset, previously "regular" observations can become anomalous in turn in the smaller dataset; it is not clear when that runaway train will stop.

In the early stages of anomaly detection, **simple data analyses** (such as descriptive statistics, 1- and 2-way tables, and traditional visualizations) may be performed to help **identify anomalous observations**, or to obtain **insights about the data**, which could eventually lead to modifications of the analysis plan.[6]

6: Which, by the way, should always be seen as a welcomed development.

How are outliers *actually* detected? Most methods come in one of two flavours: **supervised** and **unsupervised** (we will discuss those in detail in later sections).

Supervised learning (SL) methods use a historical record of **labeled** (that is to say, previously identified) anomalous observations to build a **predictive classification or regression model** which estimates the probability that a unit is anomalous; domain expertise is required to tag the data.

Since anomalies are typically **infrequent**, these models often also have to accommodate the **rare occurrence** (or class imbalance) problem.

Supervised models are built to minimize a **cost function**; in default settings, it is often the case that the mis-classification cost is assumed to be symmetrical, which can lead to technically correct but useless solutions.

For instance, the vast majority (99.999+%) of air passengers emphatically do not bring weapons with them on flights; a model that predicts that no passenger is attempting to smuggle a weapon on board a flight would be 99.999+% accurate, but it would miss the point completely.

For the **security agency**, the cost of wrongly thinking that a passenger:

- is smuggling a weapon $\Longrightarrow$ cost of a single search;
- is NOT smuggling a weapon $\Longrightarrow$ catastrophe (potentially).

The wrongly targeted individuals may have a ... somewhat different take on this, however, either from a societal or a personal perspective.[7]

7: Or both.

Unsupervised methods, on the other hand, use no previously labeled information (anomalous/non-anomalous) or data, and try to determine if an observation is an outlying one solely by comparing its behaviour to that of the other observations.

As an example, if all participants in a workshop except for one can view the video conference lectures, then the one individual/internet connection/computer is **anomalous** – it behaves in a manner which is different from the others.

It is **very important to note** that this **DOES NOT** mean that the different behaviour is the one we are actually interested in/searching for! In Figure 26.1, perhaps we were interested in the slightly larger red fish that swims in a different direction than the rest of the school, but perhaps we were really interested in the regular-sized teal fish that swims in the same direction as the others but that has orange eyes (can you spot it?).

Outlier Tests The following traditional methods and tests of outlier detection fall into this category:[8]

8: Note that **normality** of the data is an assumption for some of them; how robust these tests are against departures from this assumption depends on the situation.

- Perhaps the most commonly-used test is **Tukey's boxplot test**; for normally distributed data, regular observations typically lie between the **inner fences**

$$Q_1 - 1.5(Q_3 - Q_1) \quad \text{and} \quad Q_3 + 1.5(Q_3 - Q_1).$$

  Suspected outliers lie between the inner fences and their respective **outer fences**

$$Q_1 - 3(Q_3 - Q_1) \quad \text{and} \quad Q_3 + 3(Q_3 - Q_1).$$

  Points beyond the outer fences are identified as **outliers** (Q_1 and Q_3 represent the data's 1st and 3rd quartile, respectively; see Figure 26.2 (a concrete example is provided in Section 15.5).
- The **Grubbs test** is another univariate test, which takes into consideration the number of observations in the dataset:

  H_0 : no outlier in the data against H_1 : exactly one outlier in the data.

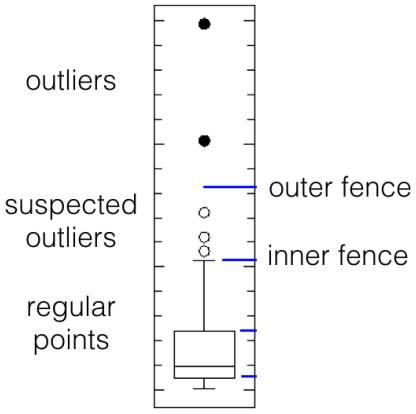


Figure 26.2: Tukey's boxplot test; suspected outliers are marked by white disks, outliers by black disks.

Let x_i be the value of feature X for the ith unit, $1 \leq i \leq n$, let $(\overline{x}, s_x)$ be the mean and standard deviation of feature X, let α be the desired significance level, and let $T(\frac{\alpha}{2n}, n)$ be the critical value of the Student t-distribution at significance $\frac{\alpha}{2n}$. The test statistic is

$$G = \frac{\max\{|x_i - \overline{x}| : i = 1, \ldots, n\}}{s_x} = \frac{|x_{i^*} - \overline{x}|}{s_x}.$$

Under H_0, G follows a special distribution with critical value

$$\ell(\alpha; n) = \frac{n-1}{\sqrt{n}} \sqrt{\frac{T^2(\frac{\alpha}{2n}, n)}{n - 2 + T^2(\frac{\alpha}{2n}, n)}}.$$

At significance level $\alpha \in (0, 1)$, we reject the null hypothesis H_0 in favour of the alternative hypothesis that x_{i^*} is the **unique outlier along feature** if $G \geq \ell(\alpha; n)$. If we are looking for more than one outlier, it can be tempting to classify every observation $\mathbf{x}_i$ for which

$$\frac{|x_i - \overline{x}|}{s_x} \geq \ell(\alpha; n)$$

as an outlier, but this approach is **contra-indicated**.

- Other common tests include:
  - the **Mahalanobis distance**, which is linked to the leverage of an observation (a measure of influence), can also be used to find multi-dimensional outliers, when all relationships are linear (or nearly linear);
  - the **Tietjen-Moore** test, which is used to find a specific number of outliers;
  - the **generalized extreme studentized deviate** test, if the number of outliers is unknown;
  - the **chi-square** test, when outliers affect the goodness-of-fit, as well as
  - DBSCAN and other clustering-based outlier detection methods;
  - visual outlier detection (see Section 15.5 for some simple examples).

26.1.2 Statistical Learning Framework

Fraudulent behaviour is not always easily identifiable, even after the fact. Credit card fraudsters, for instance, will try to disguise their transactions as regular and banal, rather than as outlandish; to fool human observers into confusing what is merely **plausible** with what is **probable** (or at least, **not improbable**).

At its most basic level, anomaly detection is a problem in **applied probability**: if I denotes what is known about the dataset (behaviour of individual observations, behaviour of observations as a group, anomalous/normal verdict for a number of similar observations, etc.), is

$$P(\text{observation is anomalous} \mid I) > P(\text{observation is not anomalous} \mid I)?$$

Anomaly detection models usually assume **stationarity for normal observations**, which is to say, that the underlying mechanism that generates data does not change in a substantial manner over time, or, if it does, that its rate of change (or cyclicity) is known.

A Time Series Detour For time series data, this means that it may be necessary to first perform trend and seasonality extraction.[9]

9: More information on these topics can be obtained in Chapter 11.

Example: supply chains play a crucial role in the transportation of goods from one part of the world to another. As the saying goes, "a given chain is only as strong as its weakest link" – in a multi-modal context, comparing the various transportation segments is far from an obvious endeavour.

If shipments departing Shanghai in February 2013 took two more days, on average, to arrive in Vancouver than those departing in July 2017, can it be said with any certainty that the shipping process has improved in the intervening years? Are February departures always slower to cross the Pacific Ocean? Are either of the Feb 2013 or the July 2017 performances anomalous?

The seasonal variability of performance is relevant to supply chain monitoring; the ability to quantify and account for the severity of its impact on the data is thus of great interest.

One way to tackle this problem is to produce an **index** to track container transit times. This index should depict the **reliability** and the **variability** of transit times but in such a way as to be able to allow for performance comparison between differing time periods.

To simplify the discussion, assume that the ultimate goal is to compare quarterly and/or monthly performance data, irrespective of the transit season, in order to determine how well the network is performing on the *Shanghai* $\longrightarrow$ *Port Metro Vancouver/Prince Rupert* $\longrightarrow$ *Toronto* corridor, say.

The supply chain under investigation has Shanghai as the point of origin of shipments, with Toronto as the final destination; the containers enter the country either through Vancouver or Prince Rupert. Containers leave their point of origin by boat, arrive and dwell in either of the two ports before reaching their final destination by rail.

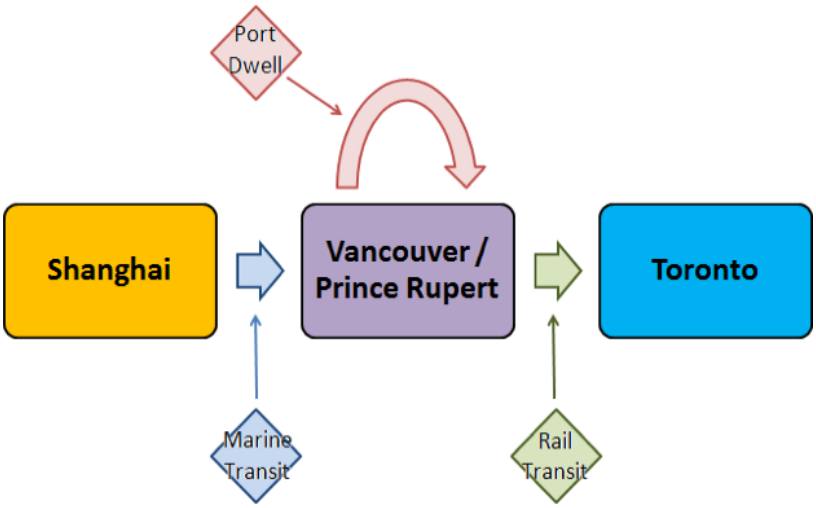


Figure 26.3: Multi-modal supply chain corridor.

For each of the three segments (Marine Transit, Port Dwell, Rail Transit), the data consists of the monthly empirical distribution of transit times, built from sub-samples (assumed to be randomly selected and fully representative) of all containers entering the appropriate segment.

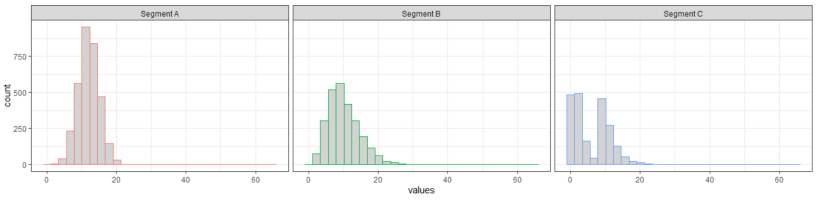
Each segment's performance is measured using **fluidity indicators**,[10] which are computed using various statistics of the transit/dwelling time distributions for each of the supply chain segments, such as:

10: In this case, compiled at a monthly scale.

Reliability Indicator (RI) the ratio of the 95th percentile to the 5th percentile of transit/dwelling times (a high RI indicates high volatility, whereas a low RI (≈ 1) indicates a reliable corridor);

Buffer Index (BI) the ratio of the positive difference between the 95th percentile and the mean, to the mean. A small BI (≈ 0) indicates only slight variability in the upper (longer) transit/dwelling times; a large BI indicates that the variability of the longer transit/dwelling times is high, and that outliers might be found in that domain;

Coefficient of Variation (CV) the ratio of the standard deviation of transit/dwelling times to the mean transit/dwelling time.



Segmt	Freq	Mean	SD	C05	C95	RI	BI	CV
A	3286	12.10	3.33	7.06	17.00	2.41	0.41	0.27
B	2594	10.09	4.43	3.88	18.20	4.69	0.80	0.44
C	2142	5.96	5.08	0.19	14.40	77.12	1.41	0.85

Figure 26.4: Illustration of how to derive the various monthly fluidity indicators.

The time series of monthly indicators (which are derived from the monthly transit/dwelling time distributions in each segment) are then **decomposed** into their:

- **trend**;
- **seasonal component** (seasonality, trading-day, moving-holiday);

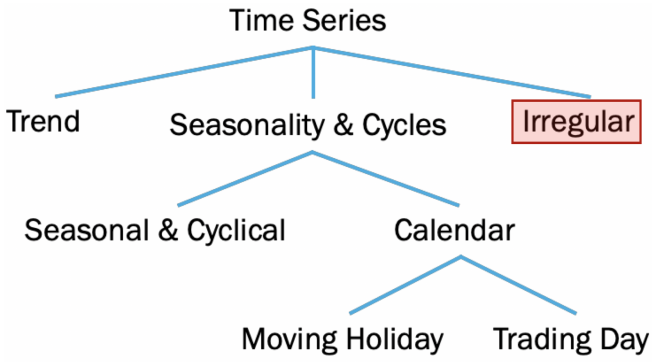


Figure 26.5: Conceptual time series decomposition; potential anomalous behaviour should be searched for in the irregular component.

- **irregular component**.

The trend and the seasonal components provide the **expected behaviour** of the indicator time series;ˆ[Before carrying out seasonal adjustment, it is important to identify and pre-adjust for structural breaks (using the Chow test, for instance), as their presence can give rise to severe distortions in the estimation of the Trend and Seasonal effects.

Seasonal breaks occur when the usual seasonal activity level of a particular time reporting unit changes in subsequent years. **Trend breaks** occurs when the trend in a data series is lowered or raised for a prolonged period, either temporarily or permanently.[11] The **irregular component** arises as a consequence of supply chain **volatility**; a high irregular component at a given time point indicates a poor performance against expectations for that month, which is to say, an **anomalous observation**.

11: Sources of these breaks may come from changes in government policies, strike actions, exceptional events, inclement weather, etc.

In general, the decomposition follows a model which is

- multiplicative;
- additive, or
- pseudo-additive.

The choice of a model is driven by data behaviour and choice of assumptions; the X12 model automates some of the aspects of the decomposition, but manual intervention and diagnostics are still required.[12] The additive model, for instance, assumes that:

12: X12 is implemented in SAS and R, among other platforms.

1. the seasonal component S_t and the irregular component I_t are independent of the trend T_t;
2. the seasonal component S_t remains stable from year to year; and
3. there is no seasonal fluctuation: $\sum_{j=1}^{12} S_{t+j} = 0$.

Mathematically, the model is expressed as:

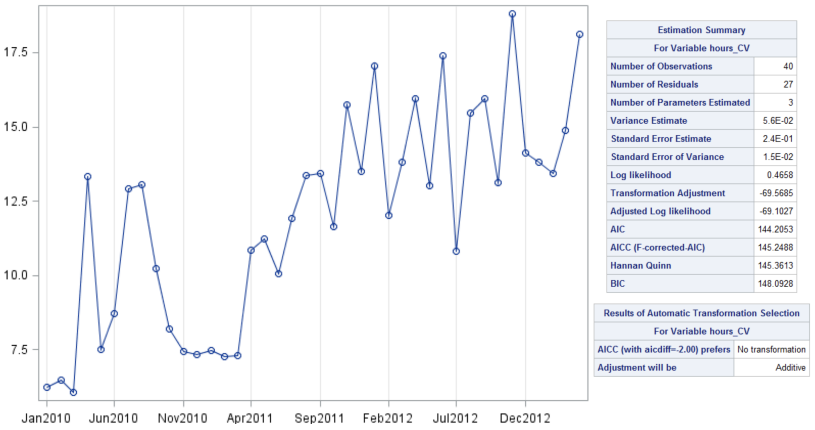
$$O_t = T_t + S_t + I_t.$$

All components share the same dimensions and units. After seasonality adjustment,the seasonality adjusted series is:

$$SA_t = O_t - S_t = T_t + I_t.$$

The multiplicative and pseudo-additive models are defined in similar ways (again, consult Chapter 11 for details).[13]

13: The simplest way to determine whether to use multiplicative or additive decomposition is by graphing the time series. If the size of the seasonal variation increases/decreases over time, multiplicative decomposition should be used. On the other hand, if the seasonal variation seems to be constant over time, additive model should be used. A pseudo-additive model should be used when the data exhibits the characteristics of the multiplicative series, but parameter values are close to zero.



Estimation Summary	
For Variable hours_CV	
Number of Observations	40
Number of Residuals	27
Number of Parameters Estimated	3
Variance Estimate	5.6E-02
Standard Error Estimate	2.4E-01
Standard Error of Variance	1.5E-02
Log likelihood	0.4658
Transformation Adjustment	-69.5685
Adjusted Log likelihood	-69.1027
AIC	144.2053
AICC (F-corrected AIC)	145.2488
Hannan Quinn	145.3613
BIC	148.0928

Results of Automatic Transformation Selection	
For Variable hours_CV	
AICC (with aicdiff=-2.00) prefers	No transformation
Adjustment will be	Additive

Figure 26.6: Marine transit CV data, from 2010 to 2013.

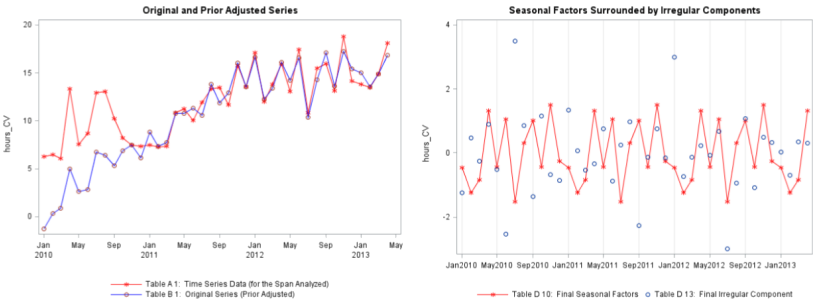


Figure 26.7: Diagnostic plot for marine transit CV data, from 2010 to 2013 (left); SI chart (right).

The data **decomposition**/preparation process is illustrated with the 40-month time series of marine transit CVs from 2010-2013, whose values are shown in Figure 26.6. The size of the peaks and troughs seems fairly constant with respect to the changing trend; the SAS implementation of X12 agrees with that assessment and suggests the additive decomposition model, with no need for further data transformations.

The diagnostic plots are shown in Figure 26.7: the CV series is prior-adjusted from the beginning until OCT2010 after the detection of a level shift. The SI (Seasonal Irregular) chart shows that there are more than one irregular component which exhibits volatility.

The **adjusted series** is shown in Figure 26.8.[14] It is on the irregular component that detection anomaly would be conducted.

14: The trend and irregular components are also shown separately for readability.

This example showcases the importance of **domain understanding** and data preparation to the anomaly detection process. As the vast majority of observations in a general problem are typically "normal", we can also view anomaly detection as a **rare occurrence learning** classification problem or as a **novelty detection** data stream problem.[15]

15: We discussed the former in Chapter 21; the latter will be tackled in Chapter 28.

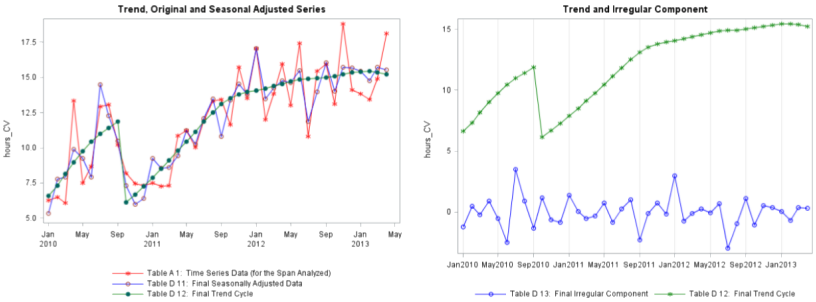


Figure 26.8: Adjusted plot for marine transit CV data, from 2010 to 2013.

While there a number of strategies that use regular classification/clustering algorithms for anomaly detection, they are rarely successful unless they are **adapted** or **modified for the anomaly detection context**.

Basic Concepts A generic system (such as the monthly transit times example) may be realized in **normal** states or in **abnormal** states. Normality, perhaps counter-intuitively, is not confined to finding the most likely state, however, as infrequently occurring states could still be normal or plausible under some interpretation of the system.

As the authors of [26] see it, a system's states are the results of processes or behaviours that follow certain natural rules and broad principles; the observations are a manifestation of these states. Data, in general, allows for inferences to be made about the underlying processes, which can then be tested or invalidated by the collection of additional data.

When the inputs are perturbed, the corresponding outputs are likely to be perturbed as well; if anomalies arise from perturbed processes, being able to identify when the process is abnormal,[16] may lead to **useful anomaly detection**.

16: That is to say, being able to capture the various normal and abnormal processes.

Any **supervised** anomaly detection algorithm requires a training set of historical labeled data (which may be costly to obtain) on which to build the prediction model, and a testing set on which to evaluate the model's performance in terms of:

- **True Positives** (TP, detected anomalies that actually arise from process abnormalities);
- **True Negatives** (TN, predicted normal observations that indeed arise from normal processes);
- **False Positives** (FP, detected anomalies corresponding to regular processes), and
- **False Negatives** (FN, predicted normal observations that are in fact the product of an abnormal process).

		Predicted Class	
		Normal	Anomaly
Actual Class	Normal	*TN*	*FP*
	Anomaly	*FN*	*TP*

Table 26.1: Confusion matrix for an anomaly detection problem.

As discussed previously, the rare occurrence problem makes optimizing for maximum **accuracy**

$$a = \frac{\text{TN} + \text{TP}}{\text{TN} + \text{TP} + \text{FN} + \text{FP}}$$

a losing strategy; instead, algorithms attempt to minimize the FP rate and the FN rate under the assumption that the cost of making a false negative error could be substantially higher than the cost of making a false positive error.

Assume that for a testing set with $\delta = \text{FN} + \text{TP}$ **true outliers**, an anomaly detection algorithm identifies $\mu = \text{FP} + \text{TP}$ **suspicious observations**, of

which $v = \text{TP}$ are **known** to be true outliers. Performance evaluation in this context is often measured using **precision** and **recall**.

Precision is the proportion of true outliers among the suspicious ones:

$$p = \frac{v}{\mu} = \frac{\text{TP}}{\text{FP} + \text{TP}};$$

when most of the observations identified by the algorithm are true outliers, $p \approx 1$; **recall** is the proportion of true outliers detected:

$$r = \frac{v}{\delta} = \frac{\text{TP}}{\text{FN} + \text{TP}};$$

when most of the true outliers are identified by the algorithm, $r \approx 1$, and the F_1-**score** is the harmonic mean of the precision and the recall:

$$F_1 = \frac{2pr}{p + r} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}.$$

One drawback of using precision, recall, and the F_1-score is that they do not incorporate TN in the evaluation process, but this is unlikely to be problematic as regular observations that are correctly seen as unsuspicious are not usually the observations of interest.[17]

Example: consider a test dataset Te with 5000 observations, 100 of which are anomalous. An algorithm which predicts all observations to be anomalous would score $a = p = 0.02$, $r = 1$, and $F_1 \approx 0.04$, whereas an algorithm that detects 10 of the true outliers would score $r = 0.1$.[18] □

17: Nevertheless, the analyst for whom the full picture is important might want to further evaluate the algorithm with the help of the **Matthews Correlation Coefficient** [37] or the **specificity** $s = \frac{\text{TN}}{\text{FP}+\text{TN}}$.

18: The other metric values would change according to the TN and FN counts.

		Predicted Class		
		Normal	Anomaly	Total
Actual Class	Normal	0	4900	4900
	Anomaly	0	100	100
	Total	0	5000	**5000**

Accuracy	0.02
Precision	0.02
Recall	1.00
F1-Score	0.04

		Predicted Class		
		Normal	Anomaly	Total
Actual Class	Normal	x	$4900 - x$	4900
	Anomaly	90	10	100
	Total	$90 + x$	$4910 - x$	**5000**

	$0 \le x \le 4900$
Accuracy	$(x + 10)/5000$
Precision	$10/(4910 - x)$
Recall	$1/10$
F1-Score	$20/(5010 - x)$

Table 26.2: Metric values for various supervised anomaly detection models.

Another supervised approach is to estimate the **relative abnormality** of various observations: it is usually quite difficult to estimate the probability that an observation $\mathbf{x}_1$ is anomalous with any certainty, but it might be possible to determine that it is more likely to be anomalous than another observation $\mathbf{x}_2$, say (denoted by $\mathbf{x}_1 \succeq \mathbf{x}_2$).

This paradigm allows the suspicious observations to be **ranked**; let $k_i \in \{1, \dots, \mu\}$ be the rank of the ith true outlier, $i \in \{1, \dots, v\}$, in the sorted list of suspicious observations

$$\mathbf{x}_1 \succeq \mathbf{x}_{k_1} \succeq \cdots \succeq \mathbf{x}_{k_i} \succeq \cdots \mathbf{x}_{k_v} \succeq \mathbf{x}_\mu;$$

the **rank power** of the algorithm is

$$RP = \frac{v(v+1)}{2\sum_{i=1}^{v} k_i}.$$

When the δ actual anomalies are ranked near the top δ suspicious ones, $\text{RP} \approx 1$. The metric is well-defined only when $\mu \ge \delta$; as with most

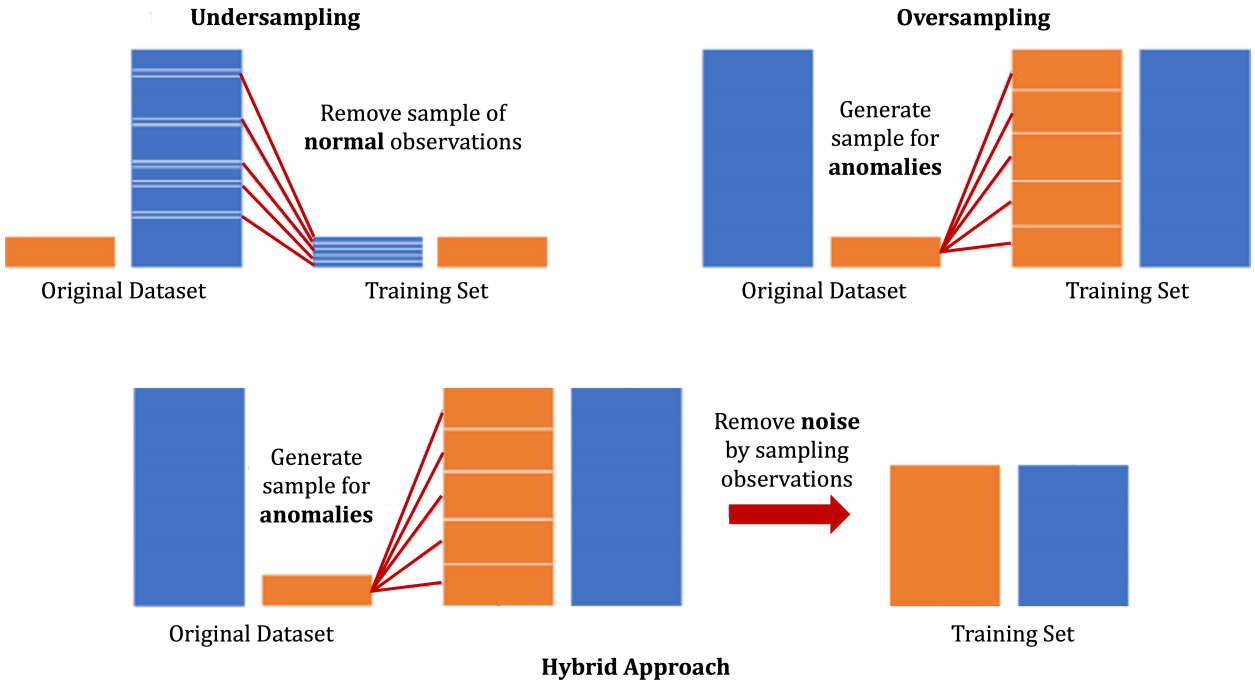


Figure 26.9: Oversampling, undersampling, and hybrid strategy for anomaly detection [22].

19: Other SL performance evaluation metrics include:

- **AUC** – the probability of ranking a randomly chosen anomaly higher than a randomly chosen normal observation (higher is better);
- **probabilistic AUC** – a calibrated version of AUC.

performance evaluation metrics, it is meaningful only in **comparison** with the performance of other algorithms.[19]

The **rare occurrence** problem can be tackled by using:

- a **manipulated training set** (oversampling, undersampling, generating artificial instances);
- **specific SL AD algorithms** (CREDOS, PN, SHRINK);
- **boosting algorithms** (SMOTEBoost, RareBoost);
- **cost-sensitive classifiers** (MetaCost, AdaCost, CSB, SSTBoost),
- etc. [21]

The rare (anomalous) class can be **oversampled** by duplicating the rare events until the data set is **balanced** (roughly the same number of anomalies and normal observations). This does not increase the overall level of information, but it will increase the misclassification cost.

The majority class (normal observations) can also be **undersampled** by randomly removing:

- "**near miss**" observations or
- observations that are "**far**" from anomalous observations.

Some loss of information has to be expected, as are "overly general" rules. Common strategies are illustrated in Figures 26.9 and Figure 26.10.

Another modern approach rests on the concept of **dimension reduction** (see Chapter 23); **autoencoders** learn a compressed representation of the data. In a sense, the **reconstruction error** measures how much information is lost in the compression.

Anomaly detection algorithms are then applied to the compressed data: we look for anomalous patterns or anomalous reconstruction errors.

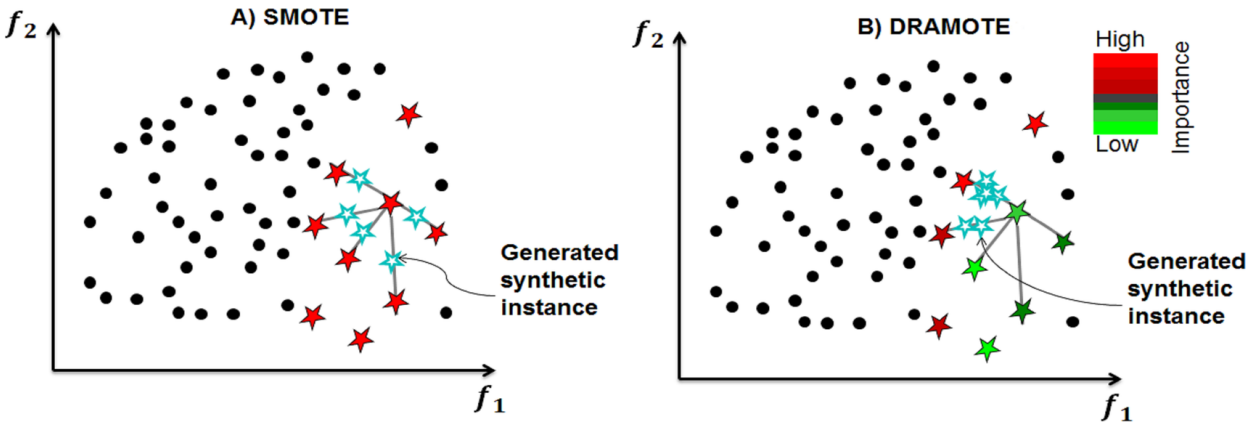


Figure 26.10: Generating artificial cases with SMOTE and DRAMOTE [33].

In the example of Figure 26.11, one observation is anomalous because its compressed representation does not follow the pattern of the other 8 observations, whereas another observation is anomalous because its reconstruction error is substantially higher than that of the other 8 observations.[20] We discuss autoencoders in more detail in Chapter 31.

20: Can you hazard a guess as to which one is which?

On the **unsupervised** front, where anomalous/normal labels are not known or used, if anomalies are those observations that are dissimilar to other observations, and if clusters represent groupings of similar observations, then observations that do not naturally fit into a cluster could be potential anomalies.[21]

21: There are a number of challenges associated to unsupervised anomaly detection, not the least of which being that most clustering algorithms do not recognize potential outliers (DBSCAN is a happy exception) and that some appropriate measure of similarity/dissimilarity of observations has to be agreed upon. Different measures may lead to different cluster assignments, as discussed in Chapter 22.

26.1.3 Motivating Example

In this chapter, we will illustrate the concepts and the algorithms of anomaly detection on an artificial dataset.

Consider a dataset of 102 observations in $\mathbb{R}^4$; the first 100 observations $\mathbf{p}_1, \ldots, \mathbf{p}_{100}$ are drawn from a multivariate $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with

$$\boldsymbol{\mu} = (1, -2, 0, 1), \quad \boldsymbol{\Sigma} = \begin{pmatrix} 1 & 0.5 & 0.7 & 0.5 \\ 0.5 & 1 & 0.95 & 0.3 \\ 0.7 & 0.95 & 1 & 0.3 \\ 0.5 & 0.3 & 0.3 & 1 \end{pmatrix}.$$

Setting-up the data

```
nobs = 100
mu = matrix(rep(c(1,-2,0,1),100), nrow=4)
Sigma = matrix(c(1, 0.5, 0.7, 0.5,
                 0.5, 1, 0.95, 0.3,
                 0.7, 0.95, 1, 0.3,
                 0.5, 0.3, 0.3, 1), nrow=4, ncol=4)
```

We use $\boldsymbol{\Sigma}$'s Cholesky decomposition to generate random observations.

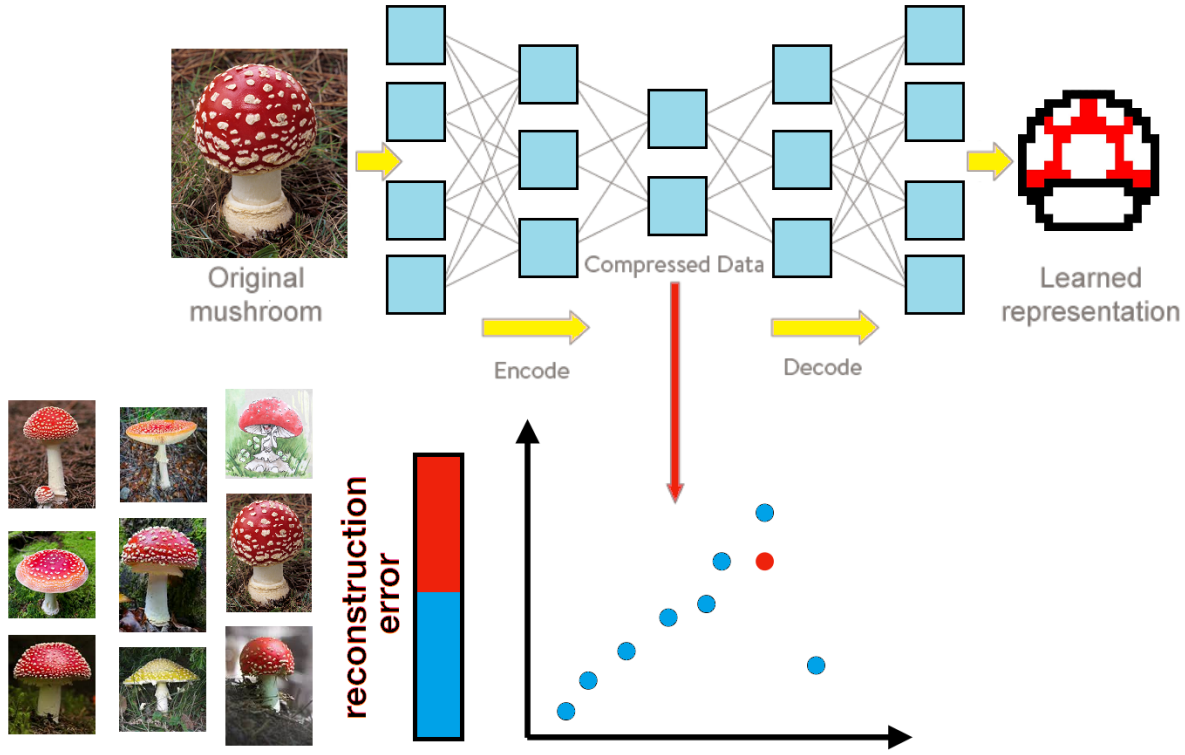


Figure 26.11: Illustration of autoencoder compression/reconstruction for anomaly detection, modified from [6].

Cholesky decomposition

```
L = chol(Sigma)
nvars = dim(L)[1]

set.seed(0) # for replicability
r = t(mu + t(L) %*% matrix(rnorm(nvars*nobs),
      nrow=nvars, ncol=nobs))
```

The summary statistics for the 100 "regular" observations are given below:

Summarizing data

```
rdata = as.data.frame(r)
names(rdata) = c('x1', 'x2', 'x3', 'x4')
summary(rdata)
```

```
       x1                 x2                 x3                 x4
 Min.   :-1.90   Min.   :-4.41   Min.   :-2.53   Min.   :-1.99
 1st Qu.: 0.38   1st Qu.:-2.65   1st Qu.:-0.62   1st Qu.: 0.34
 Median : 0.93   Median :-2.02   Median :-0.05   Median : 0.94
 Mean   : 0.94   Mean   :-1.98   Mean   : 0.01   Mean   : 0.94
 3rd Qu.: 1.46   3rd Qu.:-1.40   3rd Qu.: 0.63   3rd Qu.: 1.59
 Max.   : 3.44   Max.   : 0.52   Max.   : 2.03   Max.   : 2.81
```

We now add two observations $\mathbf{z}_1 = (1, 1, 1, 1)$ and $\mathbf{z}_4 = (4, 4, 4, 4)$ not arising from $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

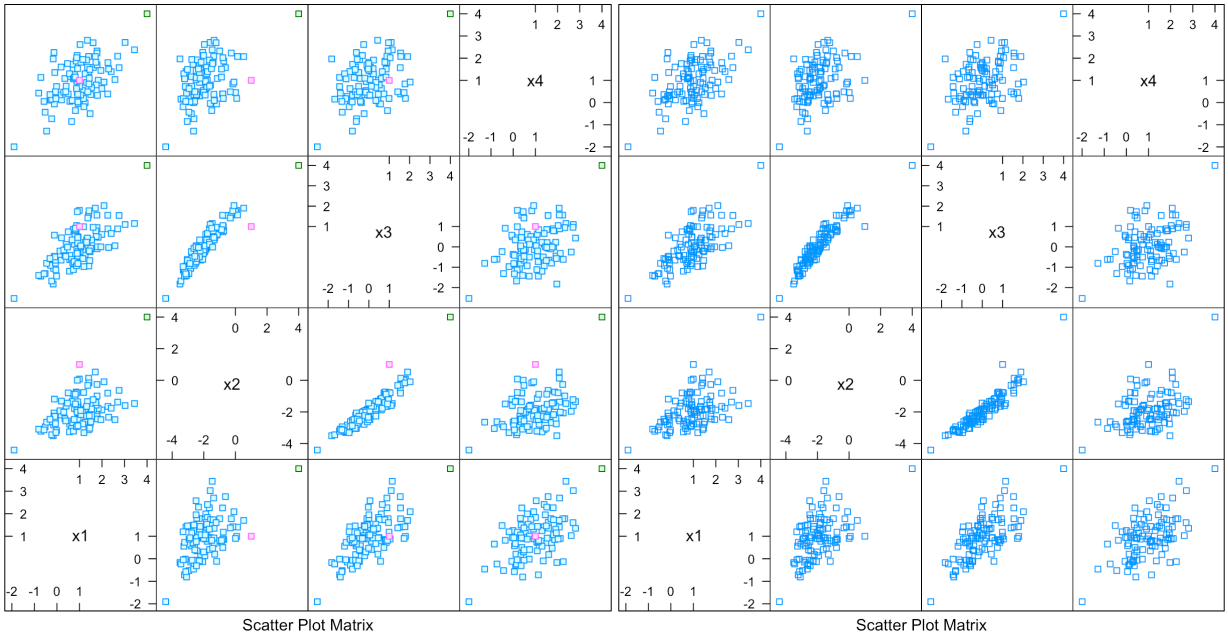
Setting-up some outliers

```
pt.1 = c(1,1,1,1)
pt.2 = c(4,4,4,4)

rdata = rbind(rdata,pt.1,pt.2)
group = c(rep(1,nobs),2,3)
rdata = cbind(rdata,group)
```

The complete dataset is displayed below, with $\mathbf{z}_1$ in pink and $\mathbf{z}_4$ in green. But since we will not usually know which observations are "regular" and which are "anomalous", let us remove the colouring.

Plotting and anonymizing the data

```
lattice::splom(rdata[,1:4], groups=group, pch=22)
lattice::splom(rdata[,1:4], pch=22)
```



Evidently, a visual inspection suggests that there are in fact 3 outliers in the dataset: the two that were specifically added as such, but a 3rd observation that was naturally outlying!

Multiple references were consulted in the preparation of this chapter, in particular [1, 26]. Other good **survey documents** include [32, 17]. Specific methods and approaches are the focus of other papers: [3, 34, 30] (**high-dimensional data**), [18] (**DOBIN**), [2] (**outlier ensembles**), [23, 12] (**isolation forest**), [10, 8] (**DBSCAN**), [7] (**LOF**), [39, 19, 29, 27, 28] (**subspace method**), [9] (**time series data**).

On the practical side, we would be remiss if we did not also mention [5], but keep in mind that there is a plethora of quality **online anomaly detection tutorials** in the programming language of your choice.

26.2 Quantitative Approaches

Cluster-based methods are not the only types of UL anomaly detection methods. Generally, they come in two flavours: **distance-based** and **density-based**.

- **Distance-based methods** include distance to all observations, distance to k nearest neighbours (kNN), average distance to kNN, median distance to kNN, etc.
- **Density-based methods** include local outlier factor (LOF), isolation forest, HDBSCAN, etc.

26.2.1 Distance Methods

In order to determine whether an observation is anomalous or not, it must be compared to a set of other observations (anomalies are relative, not absolute). In the **distance-based context**, one natural way to compare observations is to consider their distance from one another, with increasing distance from the others being increasingly suggestive of anomalous status.

This approach works both in continuous and discrete cases, as long as a **distance function** or a **pre-computed table of pair-wise distances** between observations is given.

The choice of which sets of observations to use in this comparison distinguishes the different distance-based algorithms.

Notation Let $D \subset \mathbb{R}^n$ be an n-dimensional dataset, $\mathbf{p}, \mathbf{q} \in D$, $P \subset D$ be a subset of D. Assume that $d : D \times D \to \mathbb{R}$ gives the distance between $\mathbf{p}$ and $\mathbf{q}$, written $d(\mathbf{p}, \mathbf{q})$.

An anomaly detection algorithm provides a function $a : D \to \mathbb{R}$ that describes how anomalous a given observation is. This induces an ordering on the observations of D: if $a(\mathbf{p}) < a(\mathbf{q})$ for $\mathbf{p}, \mathbf{q} \in D$, then $\mathbf{p}$ is **less anomalous** than $\mathbf{q}$.

It could be necessary to define a threshold beyond which an observation is considered anomalous; if $\alpha \in \mathbb{R}$ is such a threshold, then any $\mathbf{p} \in D$ is **absolutely anomalous** if $a(\mathbf{p}) > \alpha$.

Similarity Measures A **similarity measure** is a real-valued function that describes the similarity between two objects. A common construction is to define the similarity w between two observations $\mathbf{p}, \mathbf{q}$ as

$$w(\mathbf{p}, \mathbf{q}) = \frac{1}{1 + d(\mathbf{p}, \mathbf{q})}, \quad \text{for some distance } d,$$

so that $w \to 1$ as $d \to 0$, and $w \to 0$ as $d \to \infty$.

A similarity measure can also be constructed between probability distributions. Let X and Y be two n-dimensional random vectors of (possibly) different distribution with p.m.f./p.d.f. f_X and f_Y, respectively.

Let Ω be their shared domain. For discrete random variables, the **Hellinger distance** is defined by

$$H(X,Y) = \left(1 - \sum_{\mathbf{z}\in\Omega} \sqrt{f_X(\mathbf{z}) f_Y(\mathbf{z})}\right)^{1/2};$$

for continuous random variables, it is defined by

$$H(X,Y) = \left(1 - \int_\Omega \sqrt{f_X(\mathbf{z}) f_Y(\mathbf{z})}\, d\mathbf{z}\right)^{1/2}.$$

If $f_X = f_Y$ (or $f_X = f_Y$ almost everywhere in the continuous case, that is, except over a countable set), then

$$\sum_\Omega \sqrt{f_x f_Y} = 1 \quad \text{or} \quad \int_\Omega \sqrt{f_X f_Y}\, d\mathbf{z} = 1$$

and $H(X,Y) = 0$. The fact that $H(X,Y) \in [0,1]$ is a consequence of Cauchy's inequality, with $f_X^* = \sqrt{f_X}$ and $f_Y^* = \sqrt{f_Y}$:

$$\begin{aligned} 0 &\le \int_\Omega \sqrt{f_X f_Y}\, d\mathbf{z} = \int_\Omega f_X^* f_Y^*\, d\mathbf{z} \\ &\le \left(\int_\Omega |f_X^*|^2\, d\mathbf{z}\right)^{1/2} \left(\int_\Omega |f_Y^*|^2\, d\mathbf{z}\right)^{1/2} = \left(\int_\Omega f_X\, d\mathbf{z}\right)^{1/2} \left(\int_\Omega f_Y\, d\mathbf{z}\right)^{1/2} = 1; \end{aligned}$$

a similar argument holds for discrete random variables.

Recall that the covariance matrices $\mathbf{\Sigma}_X$ and $\boldsymbol{Sigma}_Y$ are $n \times n$-matrices whose (i,j)-th entries are the covariance between the i-th and j-th positions of X and Y, respectively. Given a collection of identically distributed samples, these covariance matrices can be estimated.

We can also consider a single observation $\mathbf{p}$ to represent a probability distribution. In that case, the Hellinger distance between that observation and any other distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\mathbf{\Sigma}$ can be studied using the framework above, using the **Mahalanobis distance**:

$$M(\mathbf{p}) = \sqrt{(\mathbf{p}-\boldsymbol{\mu})^\top \mathbf{\Sigma}^{-1} (\mathbf{p}-\boldsymbol{\mu})}.$$

Alternatively, if $\mathbf{p}$ and $\mathbf{q}$ are drawn from the same distribution with covariance $\mathbf{\Sigma}$, then the Mahalanobis distance is a dissimilarity measure between $\mathbf{p}$ and $\mathbf{q}$:

$$d_M(\mathbf{p},\mathbf{q}) = \sqrt{(\mathbf{p}-\mathbf{q})^\top \mathbf{\Sigma}^{-1} (\mathbf{p}-\mathbf{q})}.$$

Example In general, we do not know the true mean vector $\boldsymbol{\mu}$ and covariance matrix $\mathbf{\Sigma}$ from which the data could arise, and the mean vector and the covariance structure must be estimated from the data. In the example of Section 26.1, we have:

```
(mu.1 <- colMeans(rdata[,1:4]))
cov(rdata[,1:4])
```

```
         x1          x2          x3          x4
 0.96801577 -1.89096069  0.05602349  0.97433766

          x1        x2        x3        x4
x1 0.8999038 0.5690744 0.6646093 0.5033570
x2 0.5690744 1.3124241 1.0685066 0.4694309
x3 0.6646093 1.0685066 0.9921537 0.3969461
x4 0.5033570 0.4694309 0.3969461 0.9043493
```

These are distinct from the true underlying collection of parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, but close enough to be explained by **sampling variation** and because $\mathbf{z}_1, \mathbf{z}_4 \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

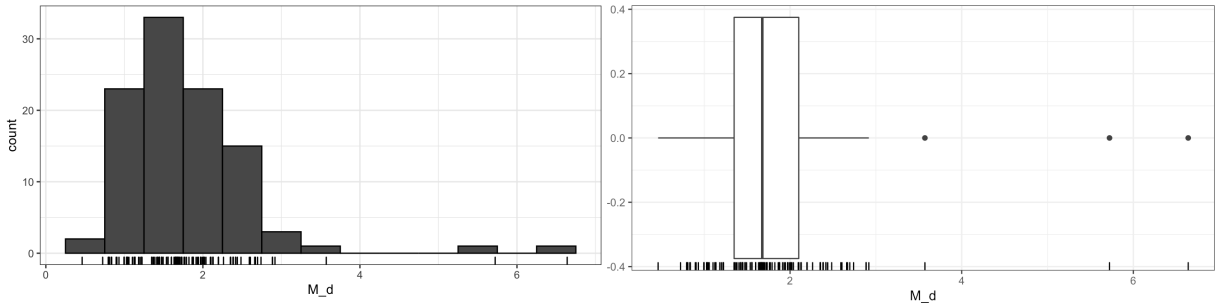
We first attempt to identify the anomalous observations by computing the Mahalanobis distance from the empirical distribution to all observations in the dataset.

```
Sigma.inv = matlib::inv(cov(rdata[,1:4]))

M_d<-vector()

for(j in 1:nrow(rdata)){
  M_d[j] <- sqrt(as.matrix(rdata[j,1:4]-mu.1) %*%
                Sigma.inv %*%
                t(as.matrix(rdata[j,1:4]-mu.1)))
}

rdata <- data.frame(rdata,M_d)
summary(M_d)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.4622  1.3479  1.6764  1.7980  2.1010  6.6393
```

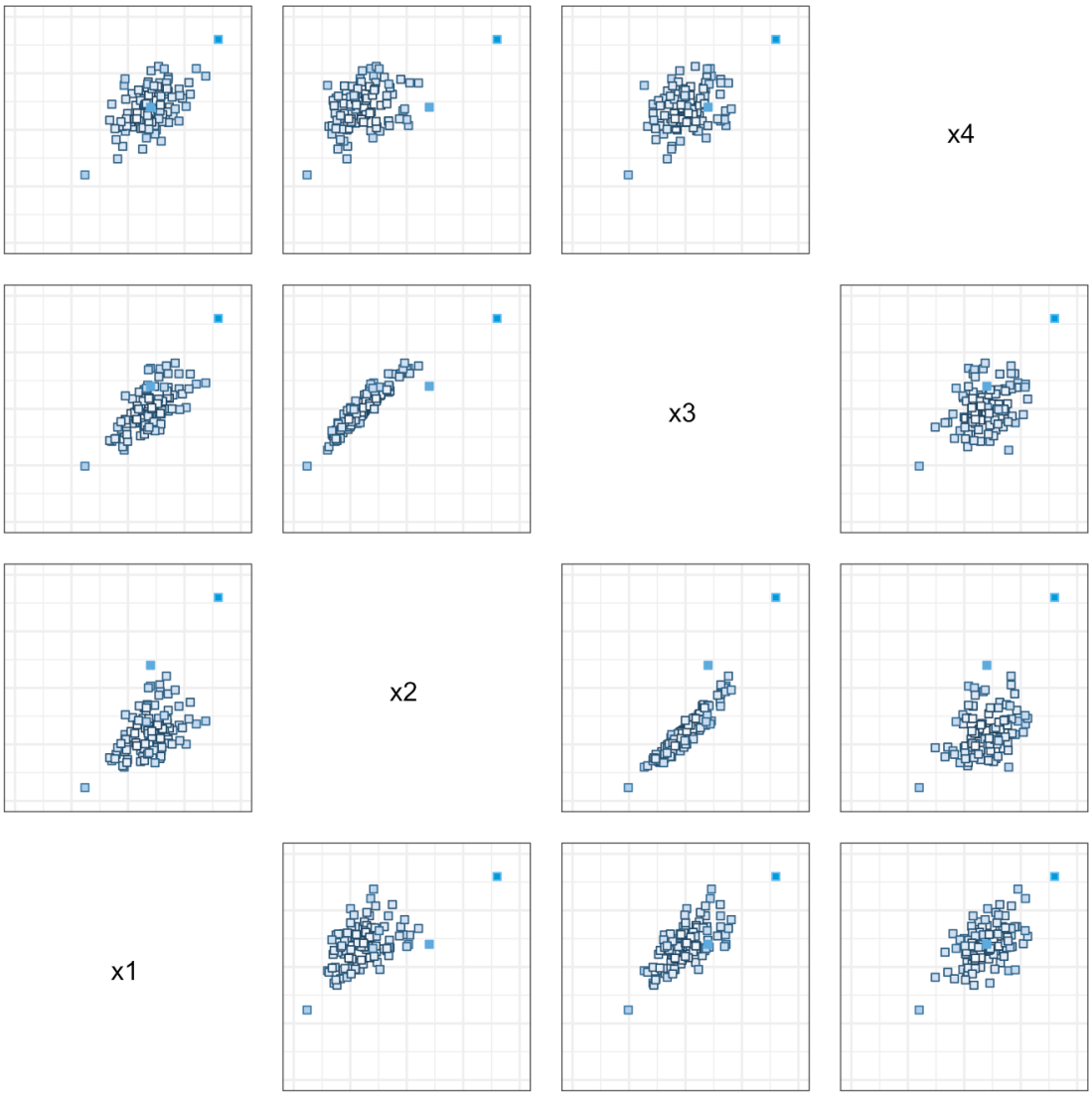
The summary suggests that there is (at least) one observation for which the Mahalanobis distance to the empirical distribution is quite high.

```
library(dplyr)   # we always assume that these
library(ggplot2) # two packages have been loaded

rdata |> ggplot(aes(x=M_d)) +
  geom_histogram(colour="black",binwidth = 0.5) +
  geom_rug() + theme_bw()

rdata |> ggplot(aes(x=M_d)) +
  geom_boxplot() + geom_rug(color="black")
```



The histogram of Mahalanobis distances shows that most observations are fairly "regular", but that two of the observations have substantially larger distances. The boxplot confirms it, but identifies a potential third outlying observation.

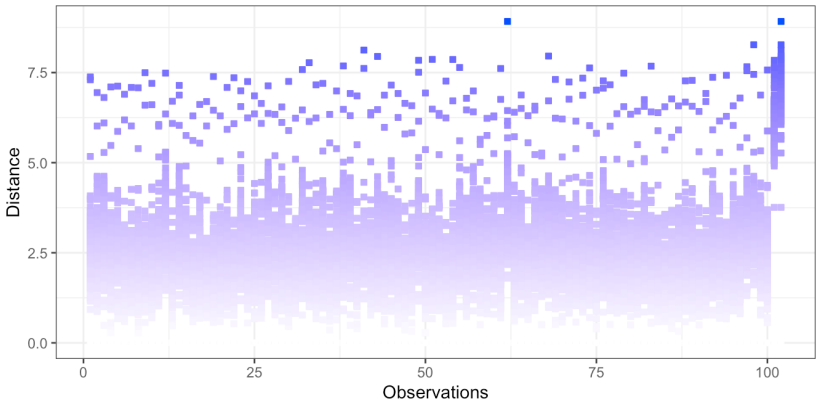
Below, we display the scatter plot matrix of the 102 observations, with colour intensity mapped to the Mahalanobis distance of the observation from the empirical distribution (the code is omitted for readability).



It certainly seems as though $\mathbf{z}_1$ and $\mathbf{z}_4$ *could* be the two anomalies.

Next, we plot the Mahalanobis distance from each observation to every other observation.

```
M_pq<-matrix(nrow=nrow(rdata), ncol=nrow(rdata))

for(j in 1:nrow(rdata)){
  for(i in 1:nrow(rdata)){
    M_pq[j,i]<-sqrt(as.matrix(rdata[j,1:4]-rdata[i,1:4]) %*%
                    Sigma.inv %*%
                    t(as.matrix(rdata[j,1:4]-rdata[i,1:4])))
  }
}

M_pq<-as.data.frame.table(M_pq)
M_pq[,1:2]<-lapply(M_pq[,1:2],as.numeric)

M_pq |> ggplot(aes(x=Var1,y=Freq)) +
  geom_point(aes(fill=Freq,colour=Freq),pch=22) +
  scale_fill_continuous(high = "#0033FF", low = "#FFFFFF") +
  scale_colour_continuous(high = "#0033FF", low = "#FFFFFF") +
  scale_x_continuous(name="Observations") +
  scale_y_continuous(name="Distance") +
  theme_bw() + theme(legend.position = "none")
```



Note the differing patterns for observations 101 and 102, as well as the diffuse cloud of points above the distance 5.0 for the other observations. There are a few other observations for which the distances to other observations seem to be larger than in a majority of the cases.

Next, we display the same distributions with the help of boxplots.

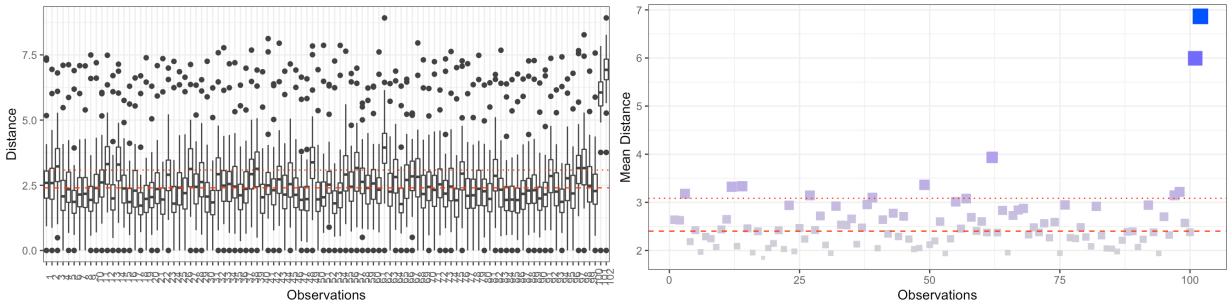
```
median.value <- M_pq |>
  group_by(Var1) |>
  summarise(meanDist=mean(Freq)) |>
  summarise(median_value=median(meanDist))

test <- M_pq |>
  group_by(Var1) |>
  summarise(meanDist=mean(Freq)) |>
  summarise(std=sd(meanDist))
```

```
med.sd = test+median.value

M_pq |> ggplot(aes(x=as.factor(Var1),y=Freq)) +
  geom_boxplot() +
  scale_x_discrete(name="Observations") +
  scale_y_continuous(name="Distance") +
  theme_bw() + theme(legend.position = "none") +
  geom_hline(yintercept=as.numeric(median.value),
             linetype = "dashed", color = "red")  +
  geom_hline(yintercept=as.numeric(med.sd),
             linetype = "dotted", color = "red")  +
  theme(axis.text.x = element_text(angle=90))
```

The long-dashed red line (see below) represents the median of all the mean distances per observation; the short-dashed red line lies 1 standard deviation above the median.

To simplify the reading of the situation, we plot only the mean distance per observation, linking the **colour intensity** and the **marker size** to the mean distance (blue corresponding to larger distances, as do larger markers).

```
M_pq |> group_by(Var1) |> summarise(meanDist=mean(Freq)) |>
  ggplot(aes(x=Var1,y=meanDist)) +
  scale_x_continuous(name="Observations") +
  scale_y_continuous(name="Mean Distance") +
  geom_point(aes(fill=meanDist,colour=meanDist,
                 size=meanDist),pch=22) +
  scale_fill_continuous(high = "#0033FF",
                        low = "#CCCCCC") +
  scale_colour_continuous(high = "#0033FF",
                          low = "#CCCCCC") +
  theme_bw() + theme(legend.position = "none") +
  geom_hline(yintercept=as.numeric(median.value),
             linetype = "dashed", color = "red")  +
  geom_hline(yintercept=as.numeric(med.sd),
             linetype = "dotted", color = "red")
```



Do any other observations strike you as potential outliers?

Similarity Measures (Reprise) If $\mathbf{\Sigma}$ is diagonal, then

$$d_M(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} \frac{(p_i - q_i)^2}{\sigma_i^2}},$$

where σ_i^2 is the variance along the i-th dimension. If Σ is the identity matrix, then we recover the **Euclidean distance**

$$d_2(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}.$$

When using the Euclidean distance in an anomaly detection context, a **linear normalization** is usually applied to each dimension so that each entry lies in the hypercube $[-1, 1]^n$. The **Minkowski distance** of order p is a generalization of the Euclidean distance:

$$d_p(\mathbf{p}, \mathbf{q}) = \left(\sum_{i=1}^{n} |p_i - q_i|^p \right)^{1/p}.$$

For $p = 2$ we recover the Euclidean distance d_2, for $p = 1$ the **Manhattan distance**

$$d_1(\mathbf{p}, \mathbf{q}) = \sum_{i=1}^{n} |p_i - q_i|,$$

and for $p = \infty$ the **supremum distance** (also called the **Chebychev distance**)

$$d_\infty(\mathbf{p}, \mathbf{q}) = \max_{i=1}^{n} |p_i - q_i|.$$

Note that the Minkowski distance d_p is only a distance function (i.e., a **metric**) when $p \geq 1$.[22]

The **Jaccard similarity** of two datasets P and Q, is defined as the size of their intersection divided by the size of their union

$$J(P, Q) = \frac{|P \cap Q|}{|P \cup Q|} = \frac{|P \cap Q|}{|P| + |Q| - |P \cap Q|}.$$

Their **Jaccard distance** is then taken to be $1 - J(P, Q)$.[23]

Finally, let $\mathbf{p}, \mathbf{q} \neq \mathbf{0}$. Recall that $\mathbf{p} \cdot \mathbf{q} = \|p\| \|q\| \cos\theta$, where θ is the angle between $\mathbf{p}$ and $\mathbf{q}$. The **cosine similarity** between $\mathbf{p}$ and $\mathbf{q}$ is the cosine of θ, which can be computed as

$$\cos\theta = \frac{p \cdot q}{\|p\| \|q\|} = \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^2} \sqrt{\sum_{i=1}^{n} q_i^2}}.$$

This value ranges between 1 and -1, with 1 attained when $\mathbf{p} = \mathbf{q}$, -1 when $\mathbf{p} = -\mathbf{q}$, and 0 when $\mathbf{p}$ and $\mathbf{q}$ are perpendicular. Armed with these concepts, we can now explore distance-based methods for anomaly detection; they will also eventually be useful for density-based anomaly detection.

22: But an exception is made for

$$d_{-\infty}(\mathbf{p}, \mathbf{q}) = \min_{i=1}^{n} |p_i - q_i|$$

to fall within the same framework.

23: This definition can be extended to compare binary vectors (i.e. vectors with entries in $\{0, 1\}$) of the same length. Given two binary vectors $\mathbf{p}$ and $\mathbf{q}$ of length n, consider an arbitrary set D of size n. Then $\mathbf{p}$ and $\mathbf{q}$ can be viewed as subsets of D: if $p_i = 1$ then $\mathbf{p}$ is said to contain the i-th element of D, while if $p_i = 0$ then it does not. Viewing $\mathbf{p}$ and $\mathbf{q}$ in this way allows us to compute their Jaccard similarity, and thus their Jaccard distance.

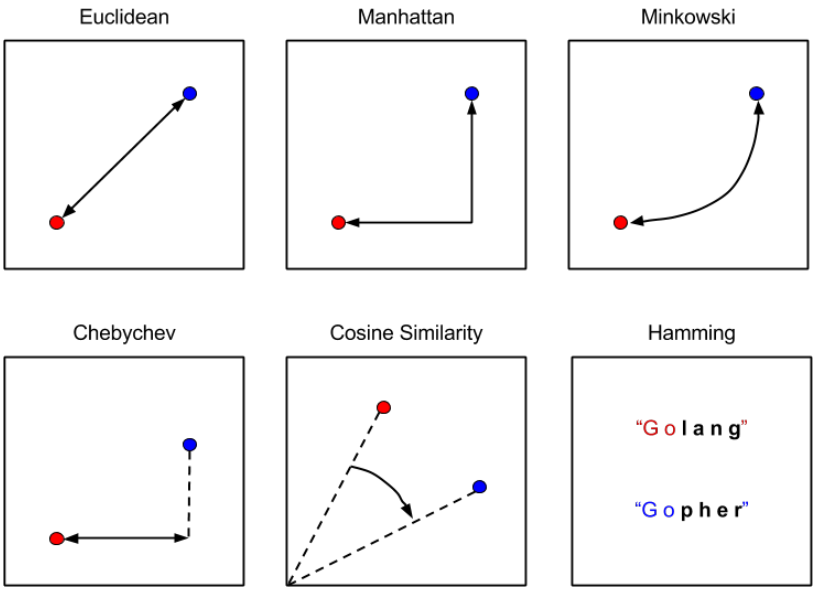


Figure 26.12: 2D visualization of various similarity metrics [36].

Distance-Based Anomaly Detection All these distance functions (similarity measures) can be used to create basic anomaly detection algorithms (the ideas can also be extended to more complex algorithms).

Given some distance function d, dataset D, and integers $k, \nu \leq |D|$, the **distance to all points** (DTAP) anomaly detection algorithm considers each observation $\mathbf{p}$ in D and adds the distance from $\mathbf{p}$ to every other observation in D, i.e.

$$a(\mathbf{p}) = \sum_{\mathbf{q} \neq \mathbf{p} \in D} d(\mathbf{q}, \mathbf{p}).$$

The ν observations with largest values for a are then said to be **anomalous according to** a. This approach often selects the most extreme observations as anomalous, which may be of limited use in practice.

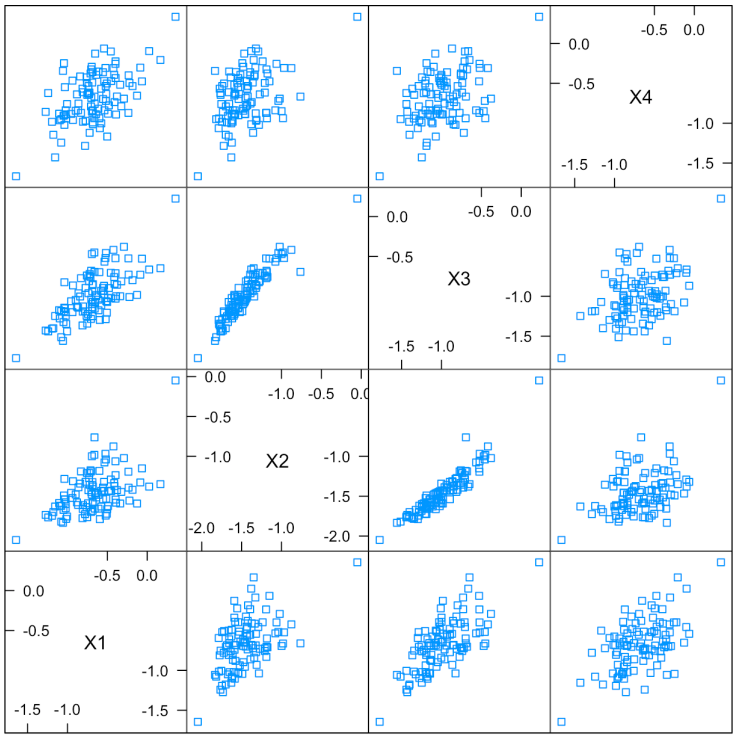
The **distance to nearest neighbour** (DTNN) algorithm defines

$$a(\mathbf{p}) = \min_{\mathbf{q} \neq \mathbf{p} \in D} d(\mathbf{q}, \mathbf{p}),$$

with a similar definition for the ν anomalous observations. The **average distance to k nearest neighbours** and **median distance to k nearest neighbours** are defined similarly.

Example: Distance to All Points We start by building the DTAP anomaly detector for the Euclidean distance (`method="euclidean"`) on the **scaled** artificial data, which is shown below.

```
rdata.scaled=data.frame(matrix(ncol = 4, nrow = nobs+2))
for(i in 1:4){
  rdata.scaled[,i] <-
    2/(max(rdata[,i]) - min(rdata[,i])) * rdata[,i] - 1
}
lattice::splom(rdata.scaled[,1:4], pch=22)
```

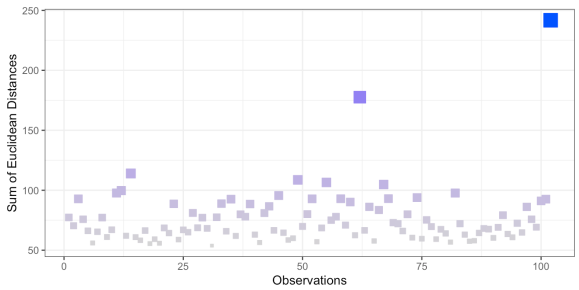


Scatter Plot Matrix

The top $v = 6$ anomalous observations are obtained as follows, with accompanying plot:

```
m.L2 <- as.matrix(dist(rdata.scaled[,1:4],
                  method="euclidean"))
adoa.L2 <- data.frame(1:(nobs+2), rowSums(m.L2))
colnames(adoa.L2) <- c("obs","dist")
adoa.L2 <- adoa.L2[order(-adoa.L2$dist),]
rownames(adoa.L2) <- NULL
head(adoa.L2)

adoa.L2 |>
  ggplot(aes(x=obs,y=dist)) +
  scale_x_continuous(name="Observations") +
  scale_y_continuous(name="Sum of Euclidean Distances") +
  geom_point(aes(fill=dist, colour=dist, size=dist),
                      pch=22) +
  scale_fill_continuous(high = "#0033FF", low = "#CCCCCC") +
  scale_colour_continuous(high = "#0033FF", low = "#CCCCCC") +
  theme_bw() + theme(legend.position = "none")
```

```
obs dist
102 241.7556
62  177.6135
14  113.9903
49  108.6464
55  106.5156
67  104.7870
```

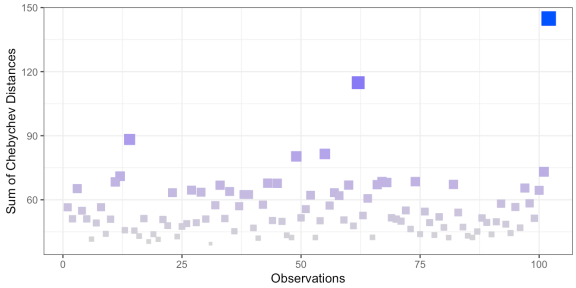


We can repeat this process for a variety of metrics.[24]

24: We do not display the code that is used for these other metrics; it can be obtained with simple modifications from the Euclidean code.

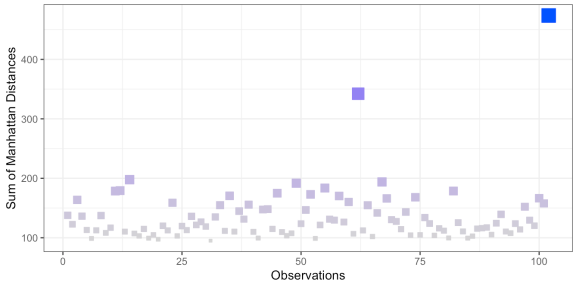
- **Chebychev** (replace `method="euclidean"` by `method="maximum"`)

```
obs dist
102 144.85387
62  114.83273
14  88.25016
55  81.47169
49  80.33274
101 73.11895
```



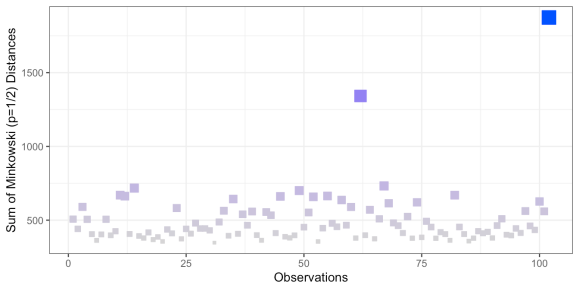
- **Manhattan** (`method="euclidean"` ↦ `method="manhattan"`)

```
obs dist
102 473.7469
62  342.2531
14  197.7493
67  193.9257
49  191.7707
55  183.6509
```



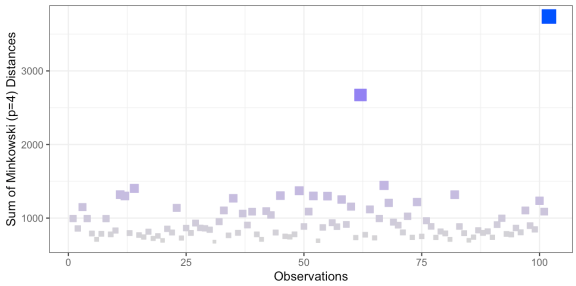
- **Minkowski**, $p = 1/2$ (`"euclidean"` ↦ `"manhattan", p=0.5`):

```
obs dist
102 1873.7488
62  1341.9020
67  731.9811
14  717.3451
49  700.5557
11  669.4313
```



- **Minkowski**, $p = 4$ (`"euclidean"` ↦ `"manhattan", p=4`):

```
obs dist
102 3738.781
62  2672.940
67  1444.177
14  1403.799
49  1372.067
11  1318.830
```



We see that while observation 102 is always the most anomalous according to DTAP, the ranking is affected by the choice of distance metric. Is this surprising?

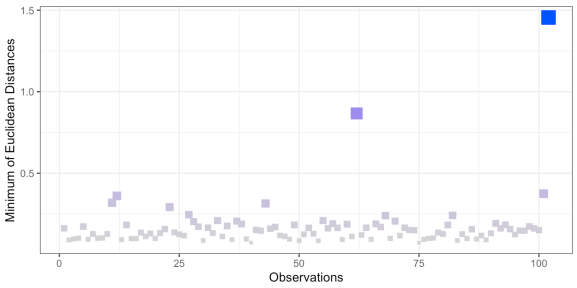
Example: Distance to Nearest Neighbour We next build the DTNN anomaly detector for the Euclidean distance, again on the **scaled** artificial data.

As before, we display the top $v = 6$ anomalous observations and the accompanying charts for 5 different metrics.

As above, we only present the code in the Euclidean case; the remaining metrics require only slight modifications. The factor 1000000 is used to create a matrix with a strongly dominant diagonal, to exclude observations being found nearest to themselves. Depending on the dataset, this factor could be reduced or may need to be increased.

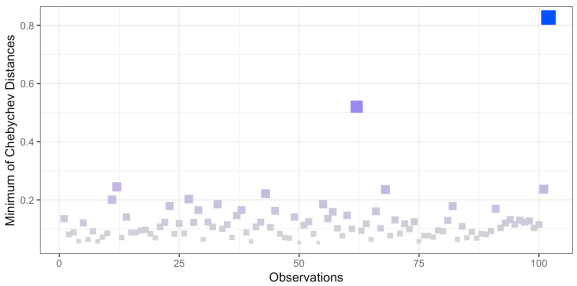
```
m.L2 <- m.L2 + 1000000*diag(nobs+2)
adoa.L2 <- data.frame(1:(nobs+2),apply(m.L2,1,min))
colnames(adoa.L2) <- c("obs","dist")
adoa.L2 <- adoa.L2[order(-adoa.L2$dist),]
rownames(adoa.L2) <- NULL
head(adoa.L2)

adoa.L2 |>
  ggplot(aes(x=obs,y=dist)) +
  scale_x_continuous(name="Observations") +
  scale_y_continuous(name="Minimum of Euclidean Distances") +
  geom_point(aes(fill=dist, colour=dist, size=dist),
                      pch=22) +
  scale_fill_continuous(high = "#0033FF",
                        low = "#CCCCCC") +
  scale_colour_continuous(high = "#0033FF",
                          low = "#CCCCCC") +
  theme_bw() + theme(legend.position = "none")
```

```
obs dist
102 1.4549129
62  0.8667313
101 0.3746565
12  0.3611265
11  0.3186967
43  0.3145305
```



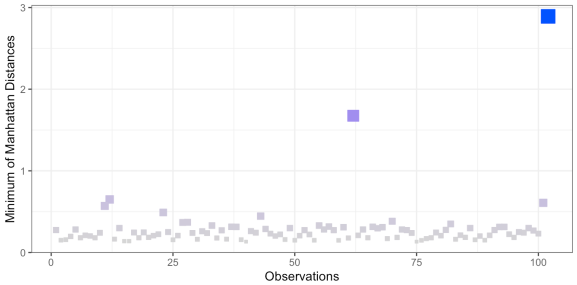
- **Chebychev** (replace `method="euclidean"` by `method="maximum"`)

```
obs dist
102 0.8269178
62  0.5203870
12  0.2449443
101 0.2368307
68  0.2356369
43  0.2214142
```



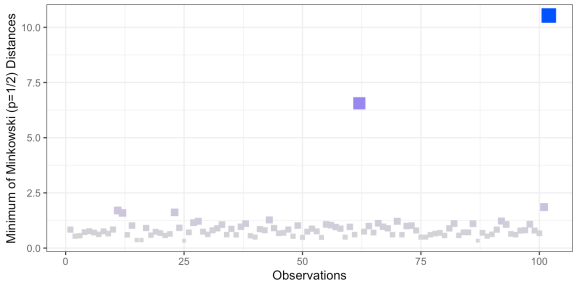
- **Manhattan** (`method="euclidean"` $\mapsto$ `method="manhattan"`)

```
obs dist
102 2.8914743
62  1.6745725
12  0.6499116
101 0.6082916
11  0.5702036
23  0.4903152
```



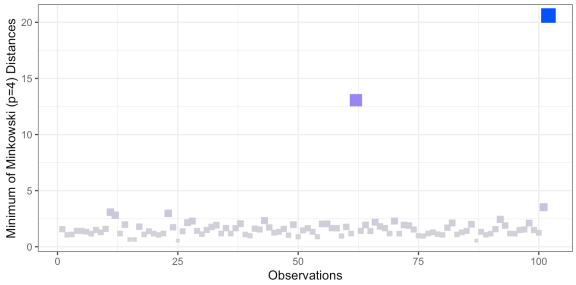
- **Minkowski**, $p = 1/2$ (`"euclidean"` $\mapsto$ `"manhattan", p=0.5`):

```
obs dist
102 10.534441
62  6.559370
101 1.857222
11  1.704864
23  1.618388
12  1.591956
```



- **Minkowski**, $p = 4$ (`"euclidean"` $\mapsto$ `"manhattan", p=4`):

```
obs dist
102 20.605432
62  13.059856
101 3.521821
11  3.081000
23  2.979090
12  2.809583
```



There are commonalities: certain observations come back repeatedly as likely anomalous observations.

Note, however, that the anomaly rankings change according to the **selected distance function** and the **choice of algorithm**; the choice of data scaling approach could also have an impact.

This is par for the course in the anomaly detection context.

26.2.2 Density Methods

Density-based approaches view observations as anomalous if they occur in **low-density regions**.

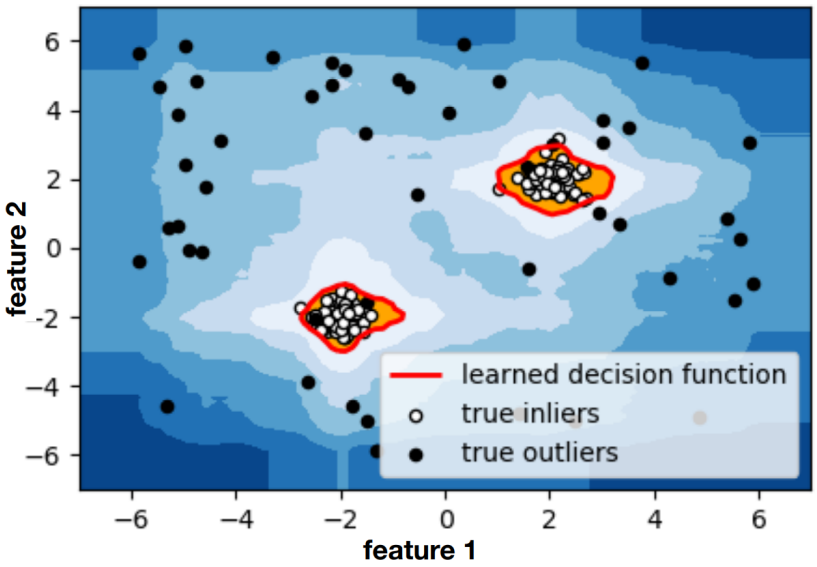


Figure 26.13: Low-density areas as outlier nurseries [6].

Density-based methods include:

- **local outlier factors**;
- **DBSCAN**, and
- **isolation forests**.

Local Outlier Factor The **Local Outlier Factor** (LOF) algorithm was proposed in 2000 by [7] (a summary can be found in Section 6.4.2 of [26]). LOF works by measuring the local deviation of each observation in a dataset from its k nearest neighbours, with a point said to be anomalous if this **deviation is large**.

A **local $k-$region** $N_k(\mathbf{p})$ around an observation $\mathbf{p}$ is defined as the k nearest neighbours of $\mathbf{p}$. The density of observations in each of their respective local $k-$neighbourhoods is estimated (the **local density**), and compared to the density of the local $k-$neighbourhoods of the observations within their own $k-$neighbourhood.

This can then be used to identify outliers that inhabit regions of lower density than their neighbours, as $\mathbf{p}$ would be in Figure 26.14.

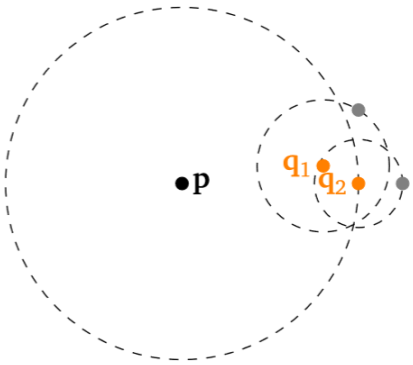


Figure 26.14: In this example, $\mathbf{p}$ has lower $k-$local density than its 2−neighbours $\mathbf{q}_1, \mathbf{q}_2$.

The formal procedure is implemented in the algorithm of Figure 26.15.

Any observation with a LOF $a_k(\mathbf{p})$ above some threshold τ is a **local outlier**, but selecting is not obvious. LOF introduces the idea of a **reachability distance**, which improves the stability of results within clusters/regions: within $N_k(\mathbf{p})$, it is

$$d_{\text{reach}}(\mathbf{p}, \mathbf{q}) = \max_{\ell}\{d(\mathbf{p}, \mathbf{q}_\ell); \mathbf{q}_\ell \in N_k(\mathbf{p})\},$$

the maximal distance to its $k-$neighbours; outside of $N_k(\mathbf{p})$, it is

$$d_{\text{reach}}(\mathbf{p}, \mathbf{q}) = d(\mathbf{p}, \mathbf{q}),$$

the actual distance.

In Figure 26.16, assuming $k = 3$, the observations $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3$ all have the same reachability distance from $\mathbf{p}$ as they are all 3-neighbours of $\mathbf{p}$, that is,

$$d_{\text{reach}}(\mathbf{p}, \mathbf{q}_1) = d_{\text{reach}}(\mathbf{p}, \mathbf{q}_2) = d_{\text{reach}}(\mathbf{p}, \mathbf{q}_3) = d(\mathbf{p}, \mathbf{q}_3).$$

The observation $\mathbf{q}_4$, on the other hand, has $d_{\text{reach}}(\mathbf{p}, \mathbf{q}_4) = d(\mathbf{p}, \mathbf{q}_4)$ as it is not a k-neighbour of $\mathbf{p}$.

Algorithm 1: Local Outlier Factor (LOF)

Input: dataset D, point $\mathbf{p} \in D$, integer k for number of nearest neighbours to consider, distance function d
Compute the distance between all points in D
for $\mathbf{p} \in D$ **do**
  for $\mathbf{q} \in D \setminus \{\mathbf{p}\}$ **do**
    Compute $d(\mathbf{p}, \mathbf{q})$
  end
  Order D by increasing distance from $\mathbf{p}$
  Set $d_k(\mathbf{p}) = d(\mathbf{p}, \mathbf{q}_k)$
end
Find the k nearest neighbours of $\mathbf{p}$
Set $N_k(\mathbf{p}) = \{\mathbf{q} \in D \setminus \{\mathbf{p}\} : d(\mathbf{p}, \mathbf{q}) \leq d_k(\mathbf{p})\}$
Define the reachability distance $d_{\text{reach}}(\mathbf{p}, \mathbf{q}) = \max\{d_k(\mathbf{q}), d(\mathbf{p}, \mathbf{q})\}$
Define the average reachability distance $\overline{d_{\text{reach}}}(\mathbf{p}) = \frac{\sum_{\mathbf{q} \in N_k(\mathbf{p})} d_{\text{reach}}(\mathbf{p}, \mathbf{q})}{|N_k(\mathbf{p})|}$
Define the local reachability density $\ell_k(\mathbf{p}) = \left(\overline{d_{\text{reach}}}(\mathbf{p})\right)^{-1}$
Compute the local outlier factor $a_k(\mathbf{p}) = \frac{\sum_{\mathbf{q} \in N_k(\mathbf{p})} \frac{\ell_k(\mathbf{q})}{\ell_k(\mathbf{p})}}{|N_k(\mathbf{p})|}$
Output: LOF $a_k(\mathbf{p})$

Figure 26.15: LOF algorithm.

Example LOF is implemented in R via the `Rlof` package; we apply it to the scaled data using the Euclidean and the Chebychev distances.

```
dist.L2 = dist(rdata.scaled[,1:4], method="euclidean")
lof <- Rlof::lof(dist.L2, k=4)
rdata.lof.L2 = data.frame(rdata.scaled[,1:4],lof)
rdata.lof.obs.L2 = data.frame(1:(nobs+2),lof)
names(rdata.lof.obs.L2) = c("obs",'lof')
rdata.lof.obs.L2 <-
        rdata.lof.obs.L2[order(-rdata.lof.obs.L2$lof),]
rownames(rdata.lof.obs.L2) <- NULL
head(rdata.lof.obs.L2)

rdata.lof.obs.L2 |>
  ggplot(aes(x=obs,y=lof)) +
  scale_x_continuous(name="Observations") +
  scale_y_continuous(name="Local outlier factor
                          (k=4, d=Euclidean)") +
  geom_point(aes(fill=lof, colour=lof, size=lof), pch=22) +
  scale_fill_continuous(high = "#0033FF", low = "#CCCCCC") +
  scale_colour_continuous(high = "#0033FF", low = "#CCCCCC") +
  theme_bw() + theme(legend.position = "none")
```

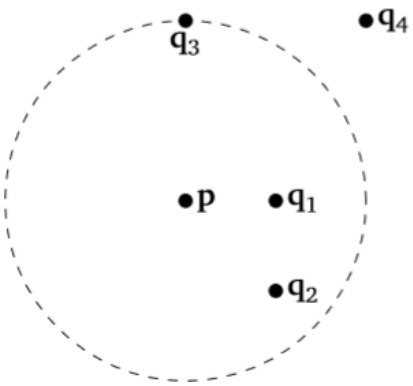
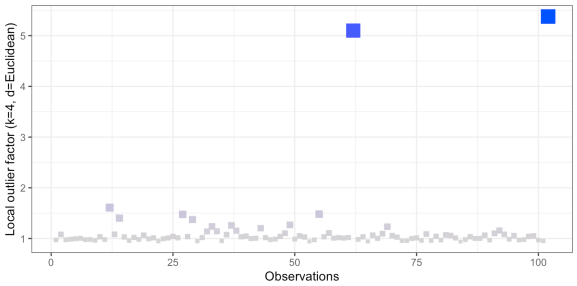


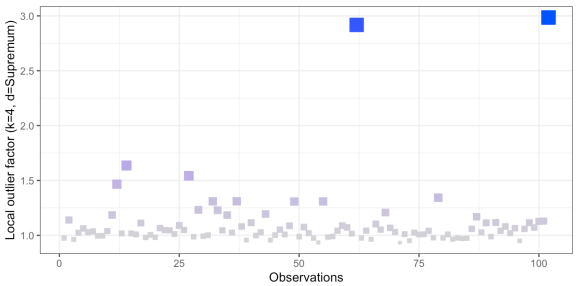
Figure 26.16: Illustration of reachability with $k = 3$.

```
obs lof
102 5.377542
62  5.100218
12  1.609463
55  1.481152
27  1.475414
14  1.402112
```




```
dist.sup = dist(rdata.scaled[,1:4], method="maximum")
lof <- Rlof::lof(dist.sup, k=4)
rdata.lof.sup = data.frame(rdata.scaled[,1:4],lof)
rdata.lof.obs.sup = data.frame(1:(nobs+2),lof)
names(rdata.lof.obs.sup) = c("obs",'lof')
rdata.lof.obs.sup <-
        rdata.lof.obs.sup[order(-rdata.lof.obs.sup$lof),]
rownames(rdata.lof.obs.sup) <- NULL
head(rdata.lof.obs.sup)

rdata.lof.obs.sup |>
  ggplot(aes(x=obs,y=lof)) +
  scale_x_continuous(name="Observations") +
  scale_y_continuous(name="Local outlier factor
                            (k=4, d=Supremum)") +
  geom_point(aes(fill=lof, colour=lof, size=lof), pch=22) +
  scale_fill_continuous(high = "#0033FF", low = "#CCCCCC") +
  scale_colour_continuous(high = "#0033FF",
                          low = "#CCCCCC") +
  theme_bw() + theme(legend.position = "none")
```

```
obs lof
102 2.984990
62  2.917587
14  1.636228
27  1.542470
12  1.466011
79  1.342456
```



DBSCAN Density-Based Spatial Clustering of Applications with Noise (DBSCAN) was proposed in 1996 by [10] (a summary can be found in Section 4.1.5 of [26], as well as in Section 22.4.1). As its name suggests, it is a density-based clustering algorithm that groups nearby observations together and labels observations that do not fall in the clusters as **anomalies/outliers**.

Hierarchical DBSCAN (HDBSCAN) [8] was introduced in 2013. It notably removes the problem of choosing the parameter for the radius of a neighbourhood by considering all "possible" radii. Further documentation can be found at [25].

In DBSCAN,

- an observation **p** is a **core point** if there is a minimum of m observations within distance r of **p**;
- an observation **q** is a **border point** (or non-core point) if it is not itself a core point but is within distance r of one, and
- an observation **o** is an **outlier** if it is neither a core point nor a border point.

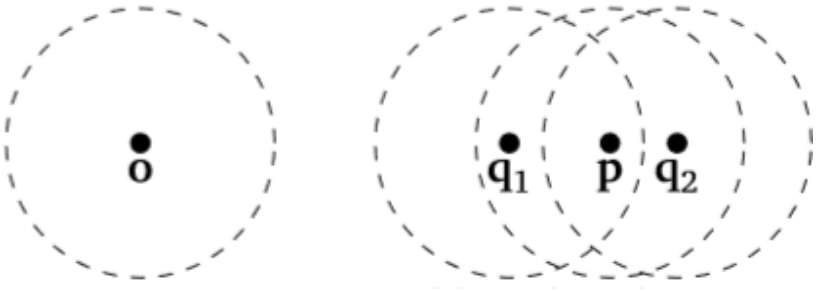


Figure 26.17: For minimum neighbourhood size $m = 2$ and the fixed radius r as displayed, **o** is an outlier, **p** is a core point, and $\mathbf{q}_1, \mathbf{q}_2$ are border points.

DBSCAN considers each observation in the dataset individually. If that observation is an outlier, then it is added to a list of outliers. Otherwise if it is a core point, then its r-neighbourhood forms the beginning of a new cluster. Each observation in this r-neighbourhood is then considered in turn, with the r-neighbourhoods of other core observations contained in the neighbourhood being **added to the cluster**.

This expansion repeats until all observations have been examined. During this step, observations that were previously labelled as outliers may be updated as they become border points in the new cluster. This process continues until every observation has either been assigned to a cluster or labelled as an outlier.

The formal procedure is implemented in the algorithm of Figure 26.18.

While DBSCAN's dual use as a **clustering algorithm** may seem irrelevant in the outlier detection setting, it is its ability to succesfully identify clusters that is crucial for labeling the remaining observations as outliers.

DBSCAN/HDSBCAN Strengths:

- the number of clusters does not need to be known beforehand (unlike in $k-$means and other clustering algorithms);
- clusters of arbitrary shape can be detected;
- when using HDBSCAN, only the parameter for the **minimum cluster size** m is required, which can be set fairly intuitively.[25]

25: This is not the case for the parameters in general clustering algorithms: if the elements of D are $n-$dimensional, the only restriction is that $m \geq n + 1$ (larger values of m allow for better noise identification).

Algorithm 2: DBSCAN

Input: dataset D, distance function d, neighbourhood radius $r > 0$, minimum number of points to be considered a cluster $m \in \mathbb{N}$
$Clusters = \{\}$
$Outliers = \{\}$
for $\mathbf{p} \in D$ **do**
  if $\mathbf{p} \in Outliers \cup (\cup_{C \in Clusters} C)$ **then**
    continue
  end
  Set $N(\mathbf{p}) = \{\mathbf{q} \in D : d(\mathbf{p}, \mathbf{q}) \leq r\}$
  if $|N(\mathbf{p})| < m$ **then**
    Add $\mathbf{p}$ to *Outliers*
    continue
  end
  else
    $Cluster = N(\mathbf{p})$
    for $\mathbf{q} \in Cluster \setminus \{\mathbf{p}\}$ **do**
      if $\mathbf{q} \in Outliers$ **then**
        Remove q from *Outliers*
      end
      else if $\mathbf{q} \in \cup_{C \in Clusters} C$ **then**
        continue
      end
      Set $N(\mathbf{q}) = \{q' \in D : d(q, q') \leq r\}$
      if $|N(\mathbf{q})| \geq m$ **then**
        Cluster = Cluster $\cup N(\mathbf{q})$
      end
    end
  end
  Add *Cluster* to *Clusters*
end
return *Outliers*
Output: a list of *outliers*

Figure 26.18: DBSCAN algorithm.

DBSCAN/HDBSCAN Limitations:

- it is not deterministic, as border points can be assigned to different clusters depending on the order in which core observations are considered – this does not affect its use as an anomaly detection algorithm, however;
- in high-dimensional spaces, the ability of any Euclidean-based distance function to distinguish near and distant observations diminishes due to the **Curse of Dimensionality**; in such spaces, it becomes ineffective (as do other clustering algorithms);

- it cannot handle differences in local densities as the radius of a neighbourhood r is fixed; this could lead to sparser clusters being labelled as outliers, or to outliers surrounding a denser cluster being included in the cluster (this issue is overcome in HDBSCAN).

Example We use the `R` implementation of DBSCAN, HDBSCAN, and OPTICS (another density-based clustering algorithm) found in the `dbscan` package; we apply various parameters to the scaled artificial data, using the Euclidean distance in all instances.[26]

26: We display all the accompanying charts in Figures 26.19 and 26.20.

```
scaled = scale(rdata[,1:4])
set.seed(1) # for replicability
```

- DSBCAN, `eps = 0.4`, `minPts = 4`

```
(db.1 <- dbscan::dbscan(scaled, eps = .4, minPts = 4))
lattice::splom(scaled, groups=db.1$cluster + 1L, pch=22)
```

```
The clustering contains 1 cluster(s) and 96 noise points.

 0  1
96  6
```

Evidently, 0.4 is too small a value for `eps` or 4 is too large a value for `minPts` (or both).

- DSBCAN, `eps = 1`, `minPts = 4`

```
(db.2 <- dbscan::dbscan(scaled, eps = 1, minPts = 4))
lattice::splom(scaled, groups=db.2$cluster + 1L, pch=22)
```

```
The clustering contains 1 cluster(s) and 6 noise points.

 0  1
 6 96
```

The results are reversed with a larger value of `eps`.

- DSBCAN, `eps = 1`, `minPts = 10`

```
(db.3 <- dbscan::dbscan(scaled, eps = 1, minPts = 10))
lattice::splom(scaled, groups=db.3$cluster + 1L, pch=22)
```

```
The clustering contains 1 cluster(s) and 24 noise points.

 0  1
24 78
```

Are the clustering results (i.e., the anomaly discovery rate) as expected? The interaction between the parameters can have unpredictable effects.

- DSBCAN, `eps = 2`, `minPts = 10`

```
(db.4 <- dbscan::dbscan(scaled, eps = 2, minPts = 10))
lattice::splom(scaled, groups=db.4$cluster + 1L, pch=22)
```

```
The clustering contains 1 cluster(s) and 2 noise points.

  0   1
  2 100
```

- HDBSCAN, `minPts = 4`

```
(hdb <- dbscan::hdbscan(scaled, minPts = 4))
lattice::splom(scaled, groups=hdb$cluster + 1L, pch=22)
```

```
The clustering contains 4 cluster(s) and 71 noise points.

 0  1  2  3  4
71 10  6  4 11
```

Note the absence of the `eps` parameter.

- OPTICS, `eps = 1`, `minPts = 4`, `eps_cl = 1`, `xi=.05`[27]

27: Read the `dbscan` package documentation for details.

```
opt <- dbscan::optics(scaled, eps = 1, minPts = 4)
(opt.1 <- dbscan::extractDBSCAN(opt, eps_cl = 1))
lattice::splom(rdata[,1:4], groups=opt.1$cluster + 1L, pch=22)
(opt.2 <- dbscan::extractXi(opt, xi = .05))
lattice::splom(scaled, groups=opt.2$cluster + 1L, pch=22)
```

```
The clustering contains 1
cluster(s) and 6 noise points.

 0  1
 6 96
```

```
The clustering contains 4
cluster(s) and 7 noise points.

0 1 2 3  4
7 4 4 11 76
```

Are there any suprises?

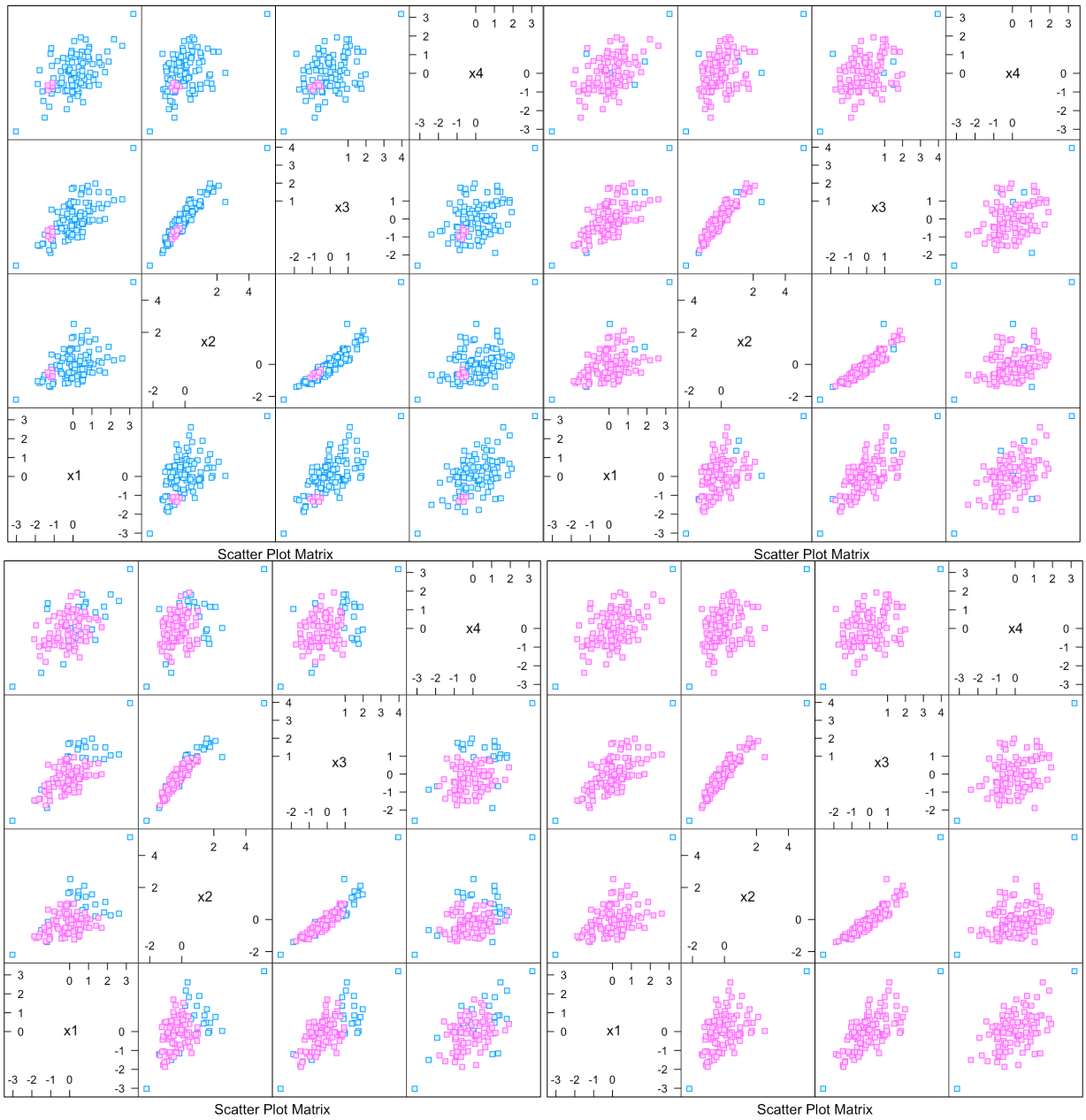


Figure 26.19: Clustering outcomes (outliers in blue): DBSCAN, top left (`eps = 0.4`, `minPts = 4`); DBSCAN, top right (`eps = 1`, `minPts = 4`); DBSCAN, bottom left (`eps = 1`, `minPts = 10`); DBSCAN, bottom right (`eps = 2`, `minPts = 10`).

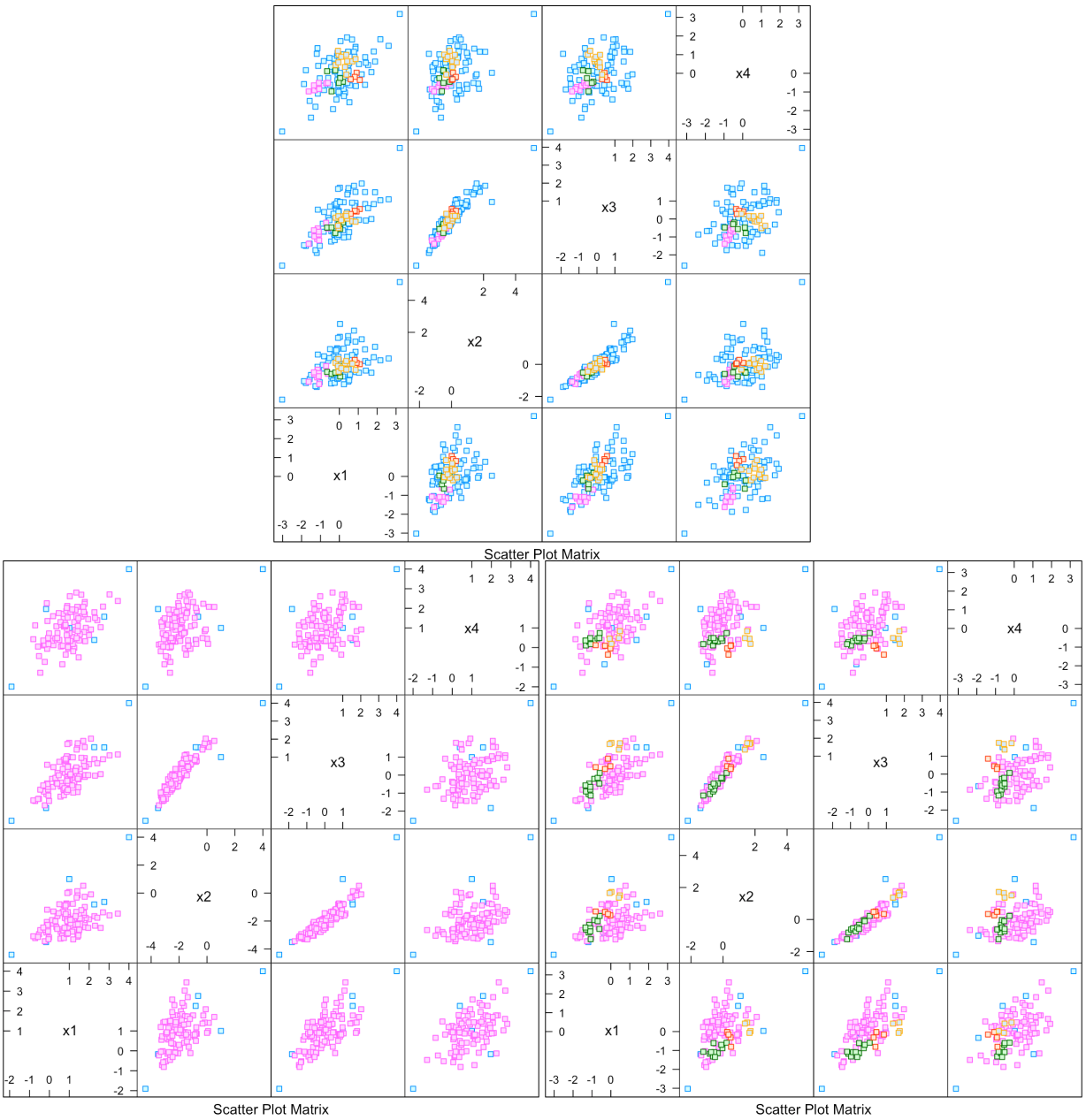


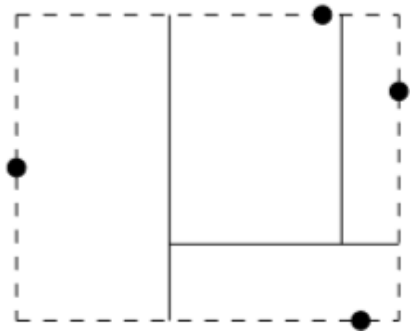
Figure 26.20: Clustering outcomes (outliers in blue): HDBSCAN, top (`minPts = 4`); OPTICS, bottom left (`eps = 0.4`, `minPts = 4`, `eps_cl = 1`, `xi = 0.05`); OPTICS, bottom right (`eps = 0.4`, `minPts = 4`, `eps_cl = 1`, `xi = 0.05`).

Isolation Forest The approaches that were previously discussed first construct models of what normal observations look like, and then identify observations that do not fit this model.

The **Isolation Forest** algorithm [23] introduced in 2008 instead tries to explicitly identify outliers under the assumptions that **there are few outliers** in the data and that these outliers have **very different attributes** compared to normal (or regular) observations.

This allows the use of sampling techniques that increase algorithmic speed while decreasing memory requirements.

The algorithm attempts to **isolate** anomalous observations by **randomly** selecting an attribute and then **randomly** selecting a split between that attribute's min and max values. This recursively partitions the observations until every observation is isolated in its own partition component.



Recursive partitioning yields a binary tree called an **Isolation Tree** (IsoTree):

- the **root** of this tree is the entire dataset;
- each **node** is a subset of the observations;
- each **branch** corresponds to one of the generated partitions, and
- the **leaves** are sets containing a single isolated observation.

Each observation is then assigned a score derived from **how deep in the tree** its singleton partition appears. Observations that are shallower in the tree are easier to separate from the rest – these are likely **outliers**.

Since only shallow observations are of interest, once the height of the tree has reached a **given threshold**,[28] further construction of the tree can be stopped to decrease computational cost.

28: The expected height of a random binary tree, say.

Instead of building a tree from the entire dataset, a tree can be constructed from a subset. The location of any observation within this smaller tree can then be estimated, again saving computational and memory resources. These two improvements are detailed in the original paper [23].

The formal procedure is implemented in the algorithm of Figure 26.21.

Once a number of Isolation Trees have been randomly generated (an **Isolation Forest**), a score can be computed for each point. This is done by searching each tree for the location of a given point and noting the path length required to reach it. Once an observation's path length in each

Algorithm 3: Recursive Isolation Tree Construction: iTree(D)

Input: dataset D
if $|D| \leq 1$ **then**
  return {}
end
else
  Let $\bar{A}$ be a list of attributes in D
  Randomly select an attribute $A \in \bar{A}$
  Randomly sample a point s from $[\min_{\mathbf{q} \in D} A(\mathbf{q}), \max_{\mathbf{q} \in D} A(\mathbf{q})]$
  Return

$$\text{Node} \begin{cases} \text{LeftChild} & = \text{iTree}(\{\mathbf{q} \in D : A(\mathbf{q}) \leq s\}) \\ \text{RightChild} & = \text{iTree}(\{\mathbf{q} \in D : A(\mathbf{q}) > s\}) \\ \text{NodeValue} & = D \end{cases}$$

end
Output: Binary tree with node values that are subsets of D

Figure 26.21: Isolation Tree algorithm.

tree has been computed, the average path length is taken to be its score. In isolated forests, low scores are indicative of outlying behaviour.

The formal procedure is implemented in the algorithm of Figure 26.23.

It can be desirable to construct a **normalized** anomaly score independent of the size of the dataset. In order to do this, the expected path length of a random observation in an Isolation Tree (i.e. binary tree) must be estimated. With $|D| = n$, it can be shown that the expected length is

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n},$$

where $H(n-1)$ is the $(n-1)^{\text{th}}$ **harmonic number**, which can be approximated by $\ln(n-1) + 0.577$; $c(n)$ is then used to normalize the final anomaly score $a(\mathbf{p})$ for $\mathbf{p} \in D$, which is given by

$$\log_2 a(\mathbf{p}) = -\frac{\text{average path length to } \mathbf{p} \text{ in the Isolation Trees}}{c(n)}.$$

Thus defined, $a(\mathbf{p}) \in [0, 1]$, with $a(\mathbf{p}) \approx 1$ suggesting $\mathbf{p}$ is an **anomaly**, $a(\mathbf{p}) \leq 0.5$ suggesting $\mathbf{p}$ is a normal observation; if all observations receive a score ≈ 0.5, this suggests that there are no anomalies present.

IsoForest Strengths:

- small time and memory requirements;
- can handle high dimensional data, and
- do not need observations to have been labeled anomalies in the training set.

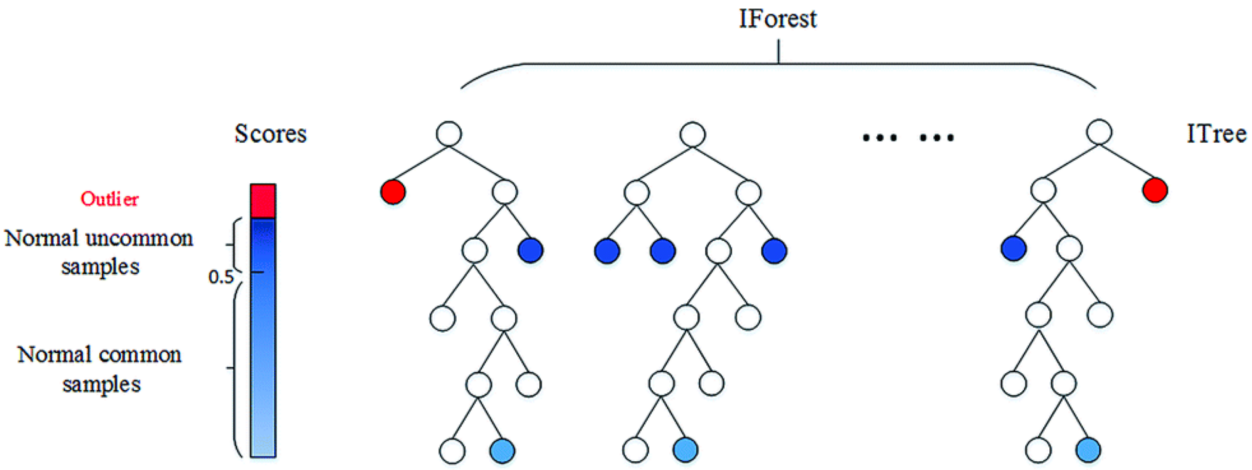


Figure 26.22: Isolation Forest schematics [6].

IsoForest Main Limitation:

- the anomaly score assigned to a given point can have high variance over multiple runs of the algorithm. The authors of [12] propose some solutions.

Example We use the `R` implementation of IsoForest found in the `solitude` package; we apply various parameters to the scaled artificial data, using the Euclidean distance in all instances. Note that this implementation uses a different scoring system, in which high scores are indicative of anomalous observations.

```
#library(solitude)
set.seed(1) # for replicability
index = 1:102
```

We initiate an isolation forest:

```
iso = solitude::isolationForest$new(
                            sample_size = length(index))
iso$fit(dataset = rbind(scaled[index,1:4],c(0,0,0,0)))
test<-iso$predict(scaled[index,1:4]) # scores for Tr data
```

The top $\nu = 6$ IsoForest anomaly scores are given below:

```
rdata.iso = data.frame(1:(nobs+2),test$anomaly_score)
names(rdata.iso) = c("obs","anomaly_score")
rdata.iso <- rdata.iso[order(-rdata.iso$anomaly_score),]
rownames(rdata.iso) = NULL
head(rdata.iso)

rdata.iso |>
  ggplot(aes(x=obs,y=anomaly_score)) +
  scale_x_continuous(name="Observations") +
  scale_y_continuous(name="IsoForest Anomaly Score") +
```

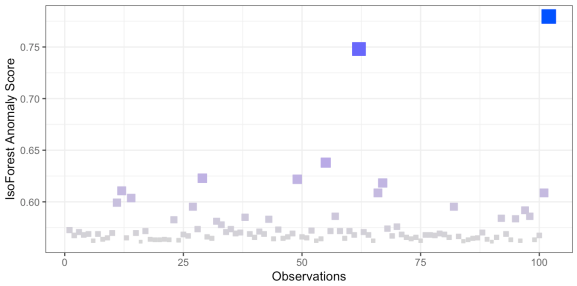
Algorithm 4: Isolation Forest

Input: dataset D, integer t number of Isolation Trees
$Forest = \{\}$
for $i = 1$ *to* t **do**
  $Tree = \text{iTree}(D)$
  Add *Tree* to *Forest*
end
for $\mathbf{p} \in D$ **do**
  $PathLengths = \{\}$
  for *Tree in Forest* **do**
    Find the path length ℓ from the root of Tree to node $\{\mathbf{p}\}$
    Add ℓ to *PathLengths*
  end
  $AveragePathLength = \frac{\sum_{\ell \in PathLengths} \ell}{t}$
  Set $a(\mathbf{p}) = 2^{-\frac{AveragePathLength}{c(|D|)}}$
end
Output: Anomaly score $a(\mathbf{p}) \in [0, 1]$ for each $\mathbf{p} \in D$

Figure 26.23: Isolation Forest algorithm.

```
  geom_point(aes(fill=anomaly_score, colour=anomaly_score,
                 size=anomaly_score), pch=22) +
  scale_fill_continuous(high = "#0033FF", low = "#CCCCCC") +
  scale_colour_continuous(high = "#0033FF", low = "#CCCCCC") +
  theme_bw() + theme(legend.position = "none")
```

```
obs anomaly_score
102 0.7795207
62  0.7480289
55  0.6379525
29  0.6228751
49  0.6218485
67  0.6182687
```



The profile of anomaly scores has a fairly distinct look (although we recognize quite a few of the usual suspects).

In general, density-based schemes are more powerful than distance-based schemes when a dataset contains **patterns with diverse characteristics**, but less effective when the patterns are of **comparable densities with the outliers** [35].

Age	Mother Tongue	Hair Colour			
		Black	Brown	Blond	Red
24-	French	11	24	12	2
	English	12	44	3	6
	Mandarin	16	2	1	0
	Arabic	9	1	0	0
	Other	11	7	13	1
25-44	French	15	32	17	2
	English	21	47	8	7
	Mandarin	23	3	1	0
	Arabic	15	2	0	2
	Other	15	16	12	6
45-64	French	7	12	2	1
	English	4	17	2	3
	Mandarin	3	1	0	0
	Arabic	3	1	0	0
	Other	6	2	1	1
65+	French	1	2	1	0
	English	3	3	0	2
	Mandarin	4	0	0	0
	Arabic	3	0	0	0
	Other	5	1	6	1

Mother Tongue	Hair Colour			
	Black	Brown	Blond	Red
French	34	70	32	5
English	40	111	13	18
Mandarin	46	6	2	0
Arabic	30	4	0	2
Other	37	26	32	9

Age	Hair Colour			
	Black	Brown	Blond	Red
24-	59	78	29	9
25-44	89	100	38	17
45-64	23	33	5	5
65+	16	6	7	3

Mother Tongue	Age			
	24-	25-44	45-64	65+
French	49	66	22	4
English	65	83	26	8
Mandarin	19	27	4	4
Arabic	10	19	4	3
Other	32	49	10	13

Hair Colour			
Black	Brown	Blond	Red
187	217	79	34
36.2%	42.0%	15.3%	6.6%

Mother Tongue				
French	English	Mandarin	Arabic	Other
141	182	54	36	104
27.3%	35.2%	10.4%	7.0%	20.1%

Age			
24-	25-44	45-64	65+
175	244	66	32
33.8%	47.2%	12.8%	6.2%

Total Number of Observations:	517	
Percentage of Levels Above:	**15%**	**25%**
Hair Colour	75%	50%
Mother Tongue	60%	60%
Age	50%	50%

Figure 26.24: 3−way, 2−way, and 1−way tables for the artificial example; the percentages of the levels above certain thresholds provide information about the spread of each of the categorical variables.

26.3 Qualitative Approaches

Non-numerical variables present new challenges when it comes to anomaly detection. A **categorical variable**[29] is one whose levels are measured on a nominal scale; examples include an object's colour, an individual's mother tongue, her favourite meal, and so on.

29: Or qualitative variable.

The **central tendency** of a categorical variable is usually given by its **mode**; measures of spread are harder to define consistently.[30]

30: One possibility is to use the proportion of levels with more than a certain percentage of the observations above a given threshold.

Consider a dataset with $n = 517$ observations and $p = 3$ predictors:

- **age** (24−, 24 − 44, 45 − 64, 65+);
- **mother tongue** (French, English, Mandarin, Arabic, Other), and
- **hair colour** (black, brown, blond, red).

The respective modes are 24 − 44, English, and brown. We can see the distribution tables in Figure 26.24, as well as the number of levels that contain more than 15% and more than 25% of the observations.

We often associate qualitative features to numerical values, but with the caveat that these **should not be interpreted as numerals**.[31]

31: If we use the code "red" = 1, "blond" = 2, "brown" = 3, and "black" = 4 to represent hair colour, we **cannot conclude** that "blond" > "red", even though 2 > 1, or that "black" − "brown" = "red", even though 4 − 3 = 1.

A categorical variable that has exactly two is called a **dichotomous feature** (or a binary variable); those with more than two levels are called **polytomous variables**.[32]

32: While we are on the topic, regression on categorical variables is called **multinomial logistic regression**.

Commonly-used distances (apart from the 0 − 1 distance and the related **Hamming distance**) typically require numerical inputs. Anomaly detection methods based on distance or on density are **not recommended** in the qualitative context.[33] Another option is to look at combinations of feature levels, but this can prove computationally expensive.

33: Unless the distance function has been modified appropriately, but that is harder to do than one may expect.

We present two specific **categorical anomaly detection** methods below: the attribute value frequency algorithm, and the greedy algorithm.

26.3.1 Attribute Value Frequency Algorithm

The **Attribute Value Frequency** (AVF) algorithm offers a fast and simple way to detect outlying observations in categorical data; it can be conducted without having to create ir which minimizes the amount of data analyses, without having to create or search through various combinations of feature levels, which reduces the overall runtime.

Intuitively, outlying observations are points which occur relatively infrequently in the (categorical) dataset; an "ideal" anomalous point is one for which **each feature value is anomalous** (or relatively infrequent). The **rarity** of an attribute level can be measured by adding the number of times the corresponding feature takes that value in the dataset.

Consider a $p-$dimensional dataset with n observations: $\{\mathbf{x}_i\}, i = 1, \ldots, n$ (each observation is a vector of p features). We write

$$\mathbf{x}_i = (x_{i,1}, \cdots, x_{i,\ell}, \cdots, x_{i,p}),$$

where $x_{i,\ell}$ is $\mathbf{x}_i$'s ℓth feature's level.

In the artificial categorial dataset presented previously above, we (perhaps) have

$$\begin{aligned} \mathbf{x}_1 &= (x_{1,1}, x_{1,2}, x_{1,3}) = (24-, \text{French}, \text{brown}) \\ &\vdots \\ \mathbf{x}_{517} &= (x_{517,1}, x_{517,2}, x_{517,3}) = (24-, \text{Mandarin}, \text{black}). \end{aligned}$$

Using the reasoning presented above, the **AVF score** is a good tool to determine whether $\mathbf{x}_i$ should be considered an outlier or not:

$$\text{AVFscore}(\mathbf{x}_i) = \frac{1}{p} \sum_{\ell=1}^{p} f(x_{i,\ell}),$$

where $f(x_{i,\ell})$ is the number of observations $\mathbf{x}$ for which the ℓth feature takes on the level $x_{i,\ell}$.

A **low** AVF score indicates that the observation is more likely to be an outlier. Since AVFscore($\mathbf{x}_i$) is essentially a sum of p positive values, it is minimized when each of the sum's term is minimized, individually.[34]

Thus, the "**ideal**" anomalous observation (in the sense described above) minimizes the AVF score; the minimal score is reached when each of the observation's features' levels occur only once in the dataset.[35]

For an integer ν, the suggested outliers are the ν observations signatures with smallest AVF score; the algorithm's complexity is $O(np)$.

The formal procedure is implemented in the algorithm of Figure 26.25.

The 10 lowest AVF scores in the artificial categorical dataset are highlighted on the next page.

34: What does this assume, if anything at all, about the features' independence.

35: Strictly speaking, the AVF score would be minimized when each of the observation's features' levels occur zero time in the dataset, but then ... the observation would not *actually* be in the dataset.

Algorithm 5: AVF

Inputs: dataset D (n observations, m features), number of anomalous observations k
while $i \leq n$ **do**
  $j = 1$
  $\text{AVFscore}(\mathbf{x}_i) = f(x_{i,j})$
  while $j \leq m$ **do**
    $\text{AVFscore}(\mathbf{x}_i) = \text{AVFscore}(\mathbf{x}_i) + f(x_{i,j})$;
    $j = j + 1$
  end
  $\text{AVFscore}(\mathbf{x}_i) = \text{Mean}(\text{AVFscore}(\mathbf{x}_i))$
  $i = i + 1$
end
Outputs: k observations with smallest AVF scores

Figure 26.25: AVF algorithm.

Age	Mother Tongue	Hair Colour			
		Black	Brown	Blond	Red
24-	French	167.7	177.7	131.7	116.7
	English	181.3	191.3	145.3	130.3
	Mandarin	138.7	148.7	102.7	87.7
	Arabic	132.7	142.7	96.7	81.7
	Other	155.3	165.3	119.3	104.3
25-44	French	190.7	200.7	154.7	139.7
	English	204.3	214.3	168.3	153.3
	Mandarin	161.7	171.7	125.7	110.7
	Arabic	155.7	165.7	119.7	104.7
	Other	178.3	188.3	142.3	127.3
45-64	French	131.3	141.3	95.3	80.3
	English	145.0	155.0	109.0	94.0
	Mandarin	102.3	112.3	66.3	51.3
	Arabic	96.3	106.3	60.3	45.3
	Other	119.0	129.0	83.0	68.0
65+	French	120.0	130.0	84.0	69.0
	English	133.7	143.7	97.7	82.7
	Mandarin	91.0	101.0	55.0	40.0
	Arabic	85.0	95.0	49.0	34.0
	Other	107.7	117.7	71.7	56.7

For instance,

$$\begin{aligned}\text{AVFscore}(24-, \text{French}, \text{blond}) &= \tfrac{1}{3}(f(24-) + f(\text{French}) + f(\text{blond})) \\ &= \tfrac{1}{3}(175 + 141 + 79) = 131.7\end{aligned}$$

For anomaly detection purposes, individual raw AVF scores are not as meaningful as relative AVF scores.

26.3.2 Greedy Algorithm

The **greedy** algorithm "greedyAlg1" is an algorithm which identifies the set of candidate anomalous observations in an efficient manner.[36] The mathematical formulation of the problem is simple – given a dataset D and a number ν of anomalous observations to identify, we solve the optimization problem

$$\text{OS} = \arg\min_{O \subseteq D}\{H(D \setminus O)\}, \quad \text{subject to } |O| = \nu,$$

36: Greedy in the sense that the algorithm picks the next step according to what is best there-and-now, and not with a long-term view.

where the **entropy** of the subset $D \setminus O$ is the sum of the entropy of each feature on $D \setminus O$ (see Section @ref(ML-CVE-dt)):

$$H(D \setminus O) = H(X_1; D \setminus O) + \cdots + H(X_m; D \setminus O)$$

and

$$H(X_\ell; D \setminus O) = -\sum_{z_\ell \in S(X_\ell; D \setminus O)} p(z_\ell) \log p(z_\ell),$$

where $S(X_\ell; D \setminus O)$ is the set of levels that the ℓth feature takes in $D \setminus O$.

The "greedyAlg1" algorithm solves the problem as follows:

1. The set of outlying and/or anomalous observations OS is initially set to be empty, and all observations of $D \setminus$ OS are identified as normal (or regular).
2. Compute $H(D \setminus \text{OS})$.
3. Scan the dataset in order to select a candidate anomalous observation: every normal observation $\mathbf{x}$ is temporarily taken out of $D \setminus$ OS to create a subset $D'_{\mathbf{x}}$, whose entropy $H(D'_{\mathbf{x}})$ is also computed.
4. The observation $\mathbf{z}$ which provides the **maximal entropy impact**, i.e. the one that minimizes

$$H(D \setminus \text{OS}) - H(D'_{\mathbf{x}}), \quad \mathbf{x} \in D \setminus \text{OS},$$

   is added to OS.
5. Repeat steps 2-4 another $\nu - 1$ times to obtain a set of ν candidate anomalous observations.

More details can be found in the source article [15]; an interesting detail is that it is **scalable** – it will also work for big datasets, provided the right framework is used.

26.4 Anomalies in High-Dimensional Data

Nowadays, real datasets are often quite **large**; in some scenarios, the observations may contain **100s or 1000s of features** (or dimensions).

Many classical methods use **proximity** (distance) concepts for anomaly detection (see Section 26.2) and can only be expected to work reasonably well in cases where the sample size n is larger than the dimension p.

The management of **high-dimensional data** ($n < p$) offers specific difficulties: in such spaces observations are often **isolated** and **scattered** (or sparse) and the notion of proximity fails to maintain its relevance.

In that case, the notion of defining significant outliers is much more **complex**: many conventional methods of detecting outliers are simply not efficient in the high-dimensional context, due to the **curse of dimensionality**. Consequently, **high-dimensional anomaly detection methods** are linked with dimension reduction and feature selection.

The remainder of this section is organized as follows: first, an attempt is made to define the concept and the challenges; then, anomaly detection techniques are discussed; finally, we end with a detailed description of ensembles and subspace methods. Our approach mainly follows those found in [1, 2, 26, 20, 4].

26.4.1 Definitions and Challenges

As we have seen previously, an anomalous observation is one that deviates or behaves differently from other the observations in the dataset, which makes us suspect that it was generated by some other mechanism [1]; such an observation would, of course, be considered to be **irregular**.

The challenges of anomaly and outlier detection in **high-dimensional data** (HDD) are due to:

- the notion of distance becoming **irrelevant** due to the curse of dimensionality (whence "the problem of detecting outliers is like finding a needle in a haystack" [20]);
- **every point** in such datasets has a tendency to be an outlier, and
- datasets become **more sparse** as the dimension of the feature space increases.

The authors of [3] consider that in order to deal properly with large datasets, detection methods should:

1. allow for **effective management** of sparse data issues;
2. provide **interpretability** of the discrepancies (i.e., how the behaviour of such observations is different);
3. allow anomaly measurements to be **compared** ("apples-to-apples");
4. consider the **local data behaviour** to determine whether an observation is abnormal or not.

26.4.2 Projection Methods

HDLSS (**high dimension, low sample size**) datasets can contain 100+ variables; the curse of dimensionality affects the efficiency of conventional anomaly/outlier detection methods.

One way to counter the problem is to **reduce the dataset's dimensionality** while preserving its essential characteristics. We have discussed such **projection methods** in Chapter 23. Let us see how one of them, **principal components analysis**, can be applied to anomaly detection.[37]

37: We take advantage of this reprise to present PCA using a different formalism.

PCA (Reprise) As we know, **principal components analysis** (PCA) aims to find a representation of the original dataset in a lower-dimensional subspace (such as a line or a plane) containing the greatest possible variation.

PCA corresponds to an **orthogonal linear transformation** of the data into a new coordinate system, such that the largest variance resulting from a scalar projection of the data is on the first coordinate (the **first principal component**), the second largest variance on the second coordinate, etc.

PCA is used in various contexts:

- as a **dimension reduction** method used during the data preprocessing step;
- as a **data visualization aid**, and, in the scenario of interest for this section,
- as an **anomaly and outlier detection** approach.

Let the dataset be represented by a numerical, centered, and scaled $n \times p$ matrix $\mathbf{X} = [\mathbf{X}_1, \cdots, \mathbf{X}_p]$ with n observations (number of rows) and p features (number of columns).

The **principal components** are linear combinations of the variables:

$$\mathbf{Y}_i = \ell_i^\top \mathbf{X} = \ell_{1,i}\mathbf{X}_1 + \cdots + \ell_{p,i}\mathbf{X}_p; \quad i = 1, \cdots, k,$$

with $k \leq p$, yielding the largest variance subjet to the constraint $\|\ell_i\| = 1$ (where $\|\cdot\|$ represents the Euclidean norm).

We can thus deduce that

$$\begin{aligned} \text{Var}(\mathbf{Y}_i) &= \text{Var}\left(\ell_i^\top \mathbf{X}\right) = \ell_i^\top \mathbf{X}^\top \mathbf{X} \ell_i = \ell_i^\top \mathbf{X}^\top \mathbf{X} \ell_i \\ \text{Cov}(\mathbf{Y}_i, \mathbf{Y}_k) &= \text{Cov}\left(\ell_i^\top \mathbf{X}, \ell_k^\top \mathbf{X}\right) = \ell_i^\top \mathbf{X}^\top \mathbf{X} \ell_k \,. \end{aligned}$$

PCA finds the **loadings vector** ℓ_1 which maximizes the variance of $\mathbf{Y}_1$:

$$\ell_1 = \underset{\|\ell_1\|=1}{\arg\max} \left\{ \ell^\top \mathbf{X}^\top \mathbf{X} \ell \right\},$$

then the uncorrelated loadings vector ℓ_2 which maximizes the variance of $\mathbf{Y}_2$:

$$\ell_2 = \underset{\|\ell_2\|=1,\, \ell_1^\top \ell_2 = 0}{\arg\max} \left\{ \ell^\top \mathbf{X}^\top \mathbf{X} \ell \right\}.$$

Similarly, the loadings vector ℓ_k is not correlated with any of the ℓ_i, $i < k$, and maximizes the variance of $\mathbf{Y}_k$:

$$\ell_k = \underset{\substack{\|\ell_k\|=1, \\ \ell_i^\top \ell_k = 0,\, \forall\, i<k}}{\arg\max} \left\{ \ell^\top \mathbf{X}^\top \mathbf{X} \ell \right\}.$$

We solve this optimization problem for all $i < k$ through the Lagrangian

$$L = \ell_k^\top \mathbf{X}^\top \mathbf{X} \ell_k - \lambda_k (\ell_k^\top \ell_k - 1) - w \ell_i^\top \ell_k.$$

The critical points are found by differentiating with respect to each of the entries of ℓ_k, λ_k and w, and setting the result to 0, which translates to:

$$\begin{aligned} \mathbf{X}^\top \mathbf{X} \ell_k &= \lambda_k \ell_k \\ \ell_k^\top \ell_k = 1 \quad &\text{and} \quad \ell_k^\top \ell_i = 0, \quad \text{for all } i < k. \end{aligned}$$

The loadings vector ℓ_k is thus the **eigenvector** of the covariance matrix $\mathbf{X}^\top \mathbf{X}$ associated to the kth largest eigenvalue. The **proportion of the variance which can be explained** by the PCA can be calculated by first noting that

$$\sum_{i=1}^{p} \text{Var}(\mathbf{Y}_i) = \sum_{i=1}^{p} \ell_i^\top \mathbf{X}^\top \mathbf{X} \ell_i = \sum_{i=1}^{p} \lambda_i \,.$$

Consequently, the proportion of the total variance explained by the ith principal component is

$$0 \leq \frac{\lambda_i}{\sum_{i=1}^{p} \lambda_i} \leq 1.$$

The quality of the PCA results is strongly dependent on the number of retained principal components, that is, on the dimension k of the subspace on which the observations are projected. There are multiple ways to select the "right" k – we will briefly present two of them.

The proportion of the total variance explained by the first k principal components is given by

$$p_k = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{p} \lambda_i}.$$

One approach is to retain k principal components, where k is the smallest value for which p_k surpasses some pre-established threshold.[38]

38: Often taken between 80% and 90%.

The **scree plot method**, on the other hand, consists in drawing the curve given by the decreasing eigenvalues (the **scree plot**), and to identify the curve's "elbows". These points correspond to principal components for which the variance decreases at a slower rate with added components. If such an elbow exists, we would retain the eigenvalues up to it.[39]

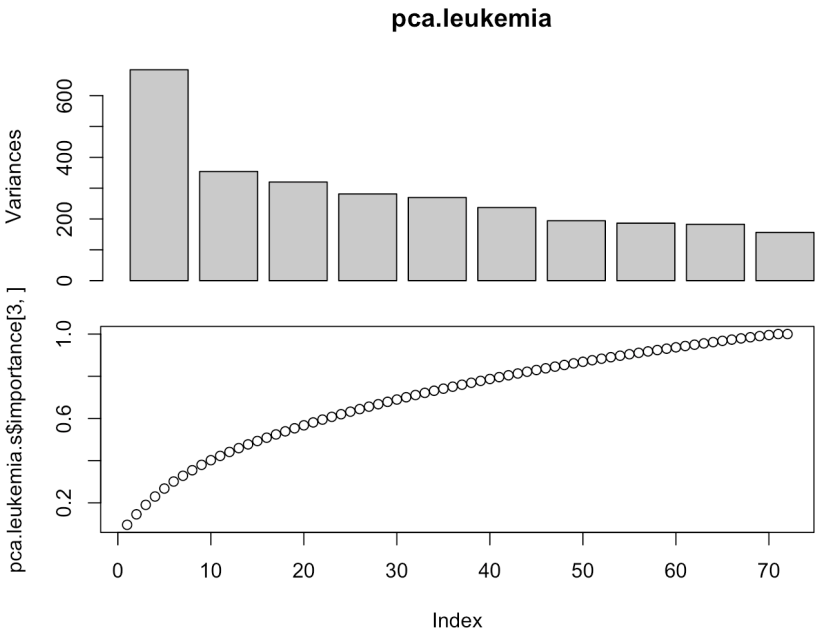
39: And thus, the corresponding principal components.

Example The leukemia_big.csv contains genetic expression measurements for $n = 72$ leukemia patients and $p = 7128$ genes [13].

```
leukemia.big <- read.csv("leukemia_big.csv")
# scale and format data
leukemia.big <- t(leukemia.big)
leukemia.big.scaled <- scale(leukemia.big)
```

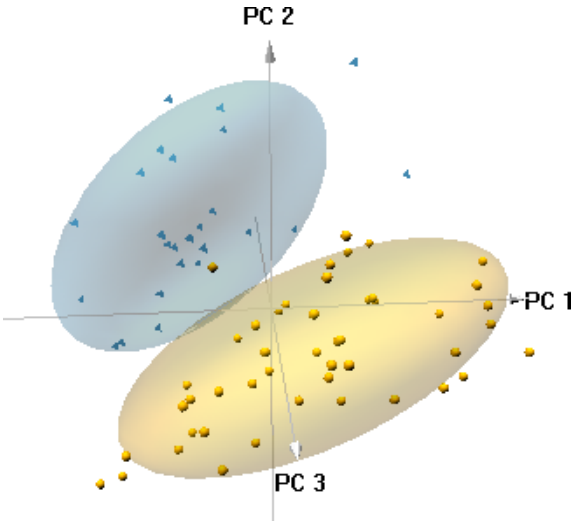
We find the PCA decomposition and display the plots.

```
pca.leukemia <- prcomp(leukemia.big.scaled)
plot(pca.leukemia)
pca.leukemia.s <- summary(pca.leukemia)
plot(pca.leukemia.s$importance[3,])
```



The scree plot suggests that only two principal components should be retained, but that does not explain an awful lot of the variation. Even keeping the first 3 principal components only explains roughly 20% of the variation.

The projection on the first 3 PC is shown below [author unknown]:



It is not obvious, however, that the methods presented here to find an optimal k are appropriate for **anomaly detection purposes**: simply put, a good k is one which **allows for good anomaly detection**.

There are other PCA-associated dimension reduction methods: independent components analysis, singular value decomposition, kernel PCA, etc. (see Chapter 23).

But what is the link with anomaly and/or outlier detection?

Once the dataset has been projected on a lower-dimensional subspace, the curse of dimensionality is (hopefully) **mitigated** – traditional methods are then applied to the **projected data**.

Dimension reduction usually leads to a loss of information, however, which can affect the accuracy of the detection procedure – especially if the presence/absence of anomalies is **not aligned** with the dataset's principal components.

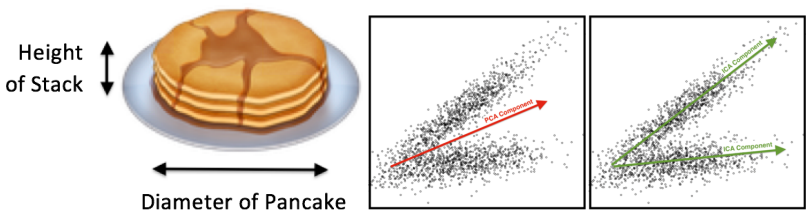
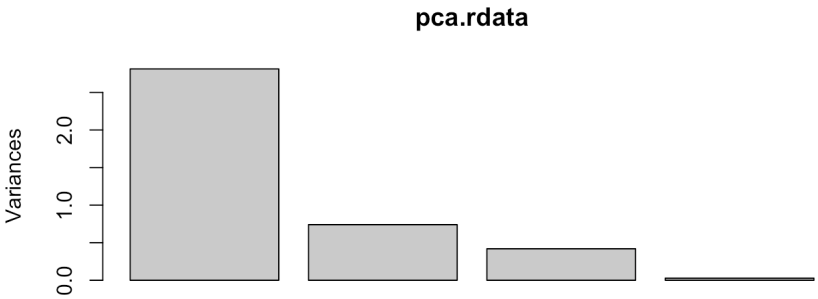


Figure 26.26: Examples of data analytical tasks that are out of alignment with PCA: pancake problem (left), in which the interesting problem might be to find the number of pancakes, but the PCA might focus on the pile's base; non-orthogonal components (right), where the data does not follow an orthogonality relation. Modified from [31].

Example Let us re-visit the artificial dataset from the earliest sections. We start by computing the PCA decomposition of the `scaled` dataset.

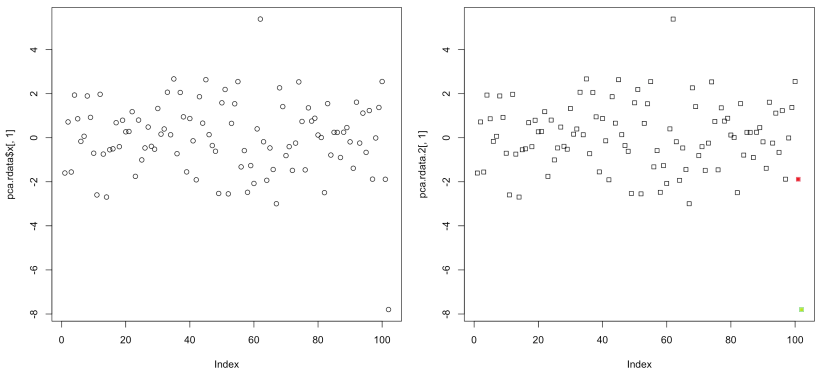
```
pca.rdata <- prcomp(scaled)
plot(pca.rdata)
```



This suggests that data is almost 1-dimensional.[40] We display the data on the first PC and highlight the "true" anomalies in the data.

40: Although the kink is on the 2nd principal component; we will revisit this soon.

```
plot(pca.rdata$x[,1])
pca.rdata.2=data.frame(pca.rdata$x[,1:4],rdata[,5])
plot(pca.rdata.2[,1], col=group, pch=22)
```



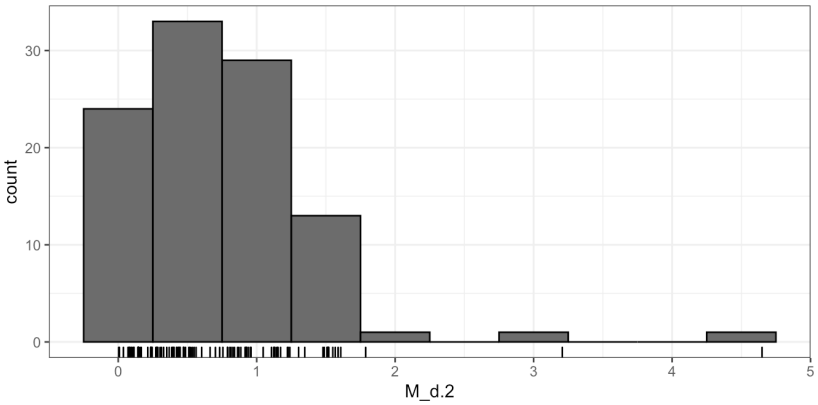
We now recreate the Mahalanobis framework on the reduced 1D data. First, we compute the empirical parameters. Next, we display the Mahalanobis distances of observations to the empirical distribution.

```
mu.2 <- mean(pca.rdata.2[,1])
Sigma.inv.2 = 1/var(pca.rdata.2[,1])

M_d.2<-vector()

for(j in 1:nrow(rdata)){
  M_d.2[j]<-sqrt(as.matrix(pca.rdata.2[j,1]-mu.2) %*%
                 Sigma.inv.2 %*%
                 t(as.matrix(pca.rdata.2[j,1]-mu.2)))
}

pca.rdata.3 <- data.frame(pca.rdata.2,M_d.2)
summary(M_d)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.4622  1.3479  1.6764  1.7980  2.1010  6.6393
```

The corresponding histogram is given below.

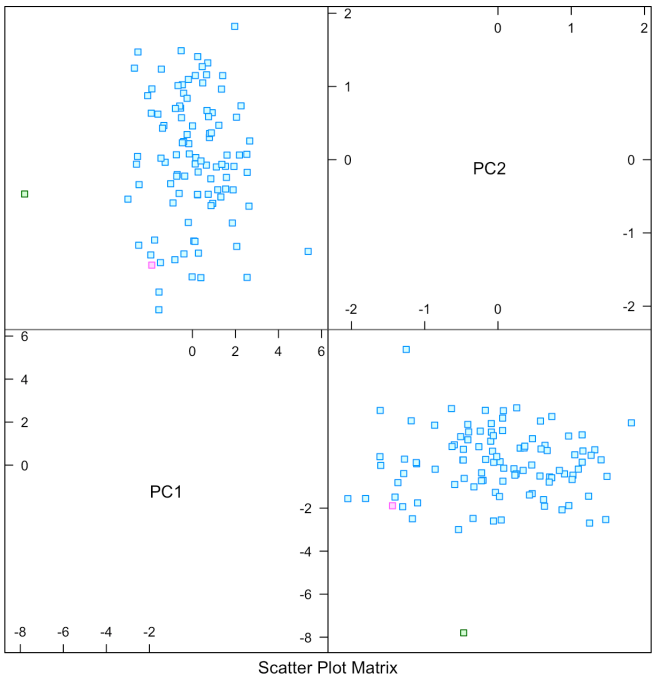
```
pca.rdata.3 |> ggplot(aes(x=M_d.2)) +
  geom_histogram(colour="black",binwidth = 0.5) +
  geom_rug() + theme_bw()
```



One outlier (observation 102, as it turns out) is still clearly visible, but observation 101 (which the Mahalanobis approach on the full unscaled dataset cleanly picks out, see Section 26.2}) gets lost in the cloud of points when we only focus on the first PC. Whatever makes the latter an outlier is **out of alignment** with PC1.

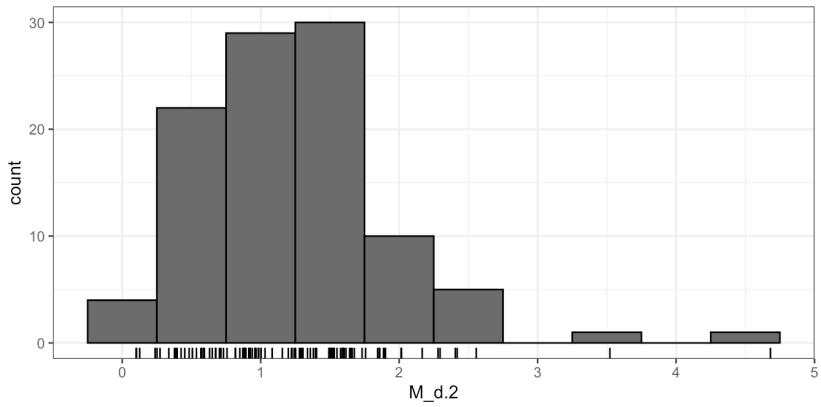
Let us use PC1 **and** PC2 to see if we can find out what is going on. The anomalies are highlighted below:

```
pca.rdata.2=data.frame(pca.rdata$x[,1:4],rdata[,5])
lattice::splom(pca.rdata.2[,1:2], groups=group, pch=22)
```



We now recreate the Mahalanobis framework on the reduced 2D data. First, we compute the empirical parameters. Next, we display the histogram of Mahalanobis distances of observations to the empirical distribution.

```
mu.2 <- colMeans(pca.rdata.2[,1:2])
Sigma.inv.2 = matlib::inv(cov(pca.rdata.2[,1:2]))

M_d.2<-vector()

for(j in 1:nrow(rdata)){
  M_d.2[j]<-sqrt(as.matrix(pca.rdata.2[j,1:2]-mu.2) %*%
                 Sigma.inv.2 %*%
                 t(as.matrix(pca.rdata.2[j,1:2]-mu.2)))
}

pca.rdata.3 <- data.frame(pca.rdata.2,M_d.2)
pca.rdata.3 |> ggplot(aes(x=M_d.2)) +
  geom_histogram(colour="black",binwidth = 0.5) +
  geom_rug() + theme_bw()
```

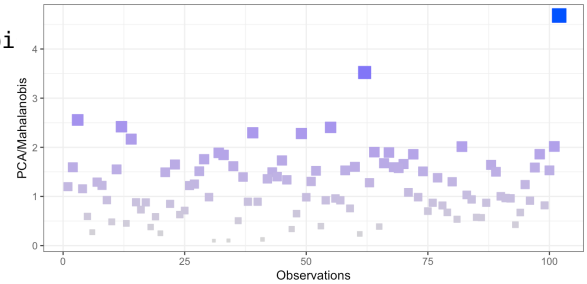


This suggests (again) that there are 2 anomalies in the data.[41]

41: The chart code is omitted – see previous examples for a baseline.

```
pca.rdata.4 = data.frame(1:(nobs+2),pca.rdata.3$M_d.2)
names(pca.rdata.4) = c("obs","PCA_Mahalanobis")
pca.rdata.4 <- pca.rdata.4[order(-pca.rdata.4$PCA_Mahalanobis),]
rownames(pca.rdata.4) <- NULL
head(pca.rdata.4)
```

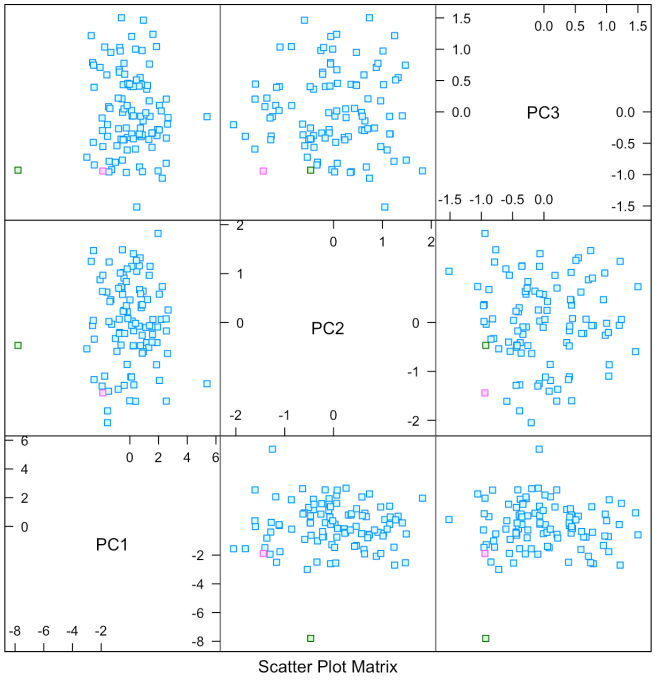
```
obs PCA_Mahalanobi
102 4.681526
62  3.521898
3   2.556683
12  2.418989
55  2.405540
39  2.295610
```



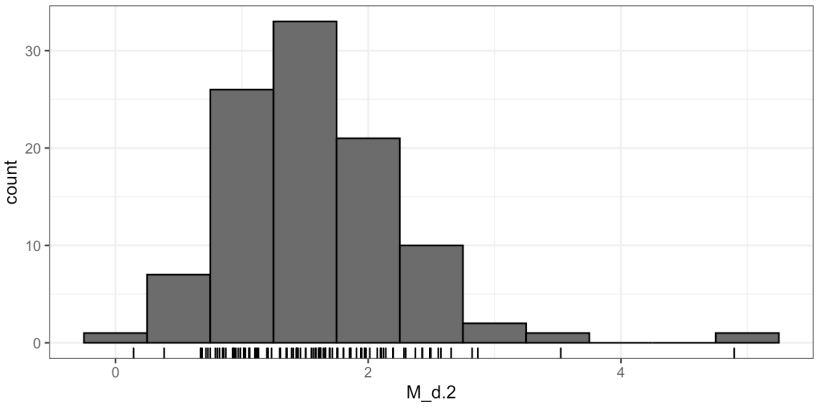
This again fails to capture observation 101 as an anomaly.

42: We omit the code as it is similar to the previous examples. Modify it as needed.

Perhaps if we use 3 PC? The anomalies are highlighted below.[42]

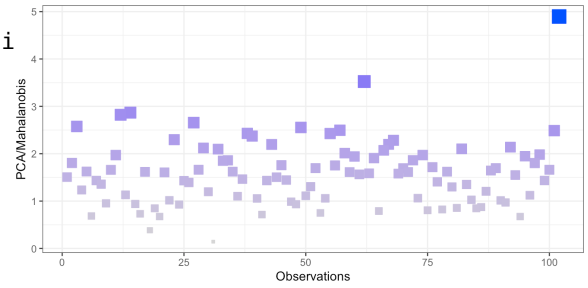


The histogram once against suggests that there are 2 anomalies in the data.



Unfortunately, the reduced 3D data once again fails to capture observation 101 as an anomaly.

```
obs PCA_Mahalanobi
102 4.896637
62  3.523881
14  2.867856
12  2.822241
27  2.656084
3   2.575923
```



Is the fact that observation 101 not captured here related to **dimension reduction**, or is it an issue related to **scaling**? Is there something that can be done to separate the two procedures? Can we get the benefits of PCA dimension reduction without having to scale the data?

Distance-Based Outlier Basis Using Neighbours As we have seen, using PCA for anomaly detection is potentially problematic: whether an observation is anomalous or not does not figure in the construction of the **principal component basis** $\{\mathrm{PC}_1, \ldots, \mathrm{PC}_k\}$ – there is not necessarily a correlation between the axes of heightened variance and the presence or absence of anomalies.

The **distance-based outlier basis using neighbours** algorithm (DOBIN) builds a basis which is better suited for the eventual detection of outlying observations. DOBIN's main concept is to search for nearest neighbours that are in fact relatively distant from one another:

1. We start by building a space $\mathbf{Y} = \{\mathbf{y}_\ell\}$ which contains $M \ll n(n+1)/2$ vectors of the form

$$\mathbf{y}_\ell = (\mathbf{x}_i - \mathbf{x}_j) \odot (\mathbf{x}_i - \mathbf{x}_j),$$

where $\odot$ is the element-by-element Hadamard multiplication, and for which the $1-$norm

$$\|\mathbf{y}_\ell\|_1 = (x_{1,1} - x_{2,1})^2 + \cdots + (x_{1,p} - x_{2,p})^2$$

is the square of the distance between $\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}$ (the selection of each of the M observation pairs is made according to a rather complex procedure which only considers $\mathbf{x}_i$ and $\mathbf{x}_j$ if they are part of one another's $k-$neighbourhood, for $k \in \{k_1, \ldots, k_2\}$); the set $\mathbf{Y}$ thus contains points for which $\|\mathbf{y}_\ell\|_1$ is relatively large, which is to say that the observations $\mathbf{x}_i$ and $\mathbf{x}_j$ are fairly distant from one another even if they are $k-$neighbours of each other;
2. we next build a basis $\{\boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_p\} \subset \mathbb{R}^p$ where each η_i is a unit vector given by a particular linear combination of points in $\mathbf{Y}$; they can be found using a Gram-Schmidt-like procedure:

$$\begin{aligned}
\mathbf{y}_{\ell_0} &= \mathbf{y}_\ell, \quad \ell = 1, \ldots, M \\
\boldsymbol{\eta}_1 &= \frac{\sum_{\ell=1}^M \mathbf{y}_\ell}{\left\|\sum_{\ell=1}^M \mathbf{y}_\ell\right\|_2} \\
\mathbf{y}_{\ell_1} &= \mathbf{y}_\ell - \langle \boldsymbol{\eta}_1 \mid \mathbf{y}_\ell \rangle, \quad \ell = 1, \ldots, M \\
\boldsymbol{\eta}_2 &= \frac{\sum_{\ell=1}^M \mathbf{y}_{\ell_1}}{\left\|\sum_{\ell=1}^M \mathbf{y}_{\ell_1}\right\|_2} \\
&\vdots \\
\mathbf{y}_{\ell_{b-1}} &= \mathbf{y}_{\ell_{b-2}} - \langle \boldsymbol{\eta}_{b-1} \mid \mathbf{y}_{\ell_{b-2}} \rangle, \quad \ell = 1, \ldots, M \\
\boldsymbol{\eta}_b &= \frac{\sum_{\ell=1}^M \mathbf{y}_{\ell_{b-1}}}{\left\|\sum_{\ell=1}^M \mathbf{y}_{\ell_{p-1}}\right\|_2},
\end{aligned}$$

for $b = 1, \ldots, p$,
3. and we transform the original dataset $\mathbf{X}$ *via* $\hat{\mathbf{X}} = \mathcal{T}(\mathbf{X})\boldsymbol{\Theta}$, where $\mathcal{T}(\mathbf{X})$ normalizes each feature of $\mathbf{X}$ according to a problem-specific scheme (Min-Max or Median-IQR, say), and

$$\boldsymbol{\Theta} = [\boldsymbol{\eta}_1 \mid \cdots \mid \boldsymbol{\eta}_p]$$

is an orthogonal $p \times p$ matrix.

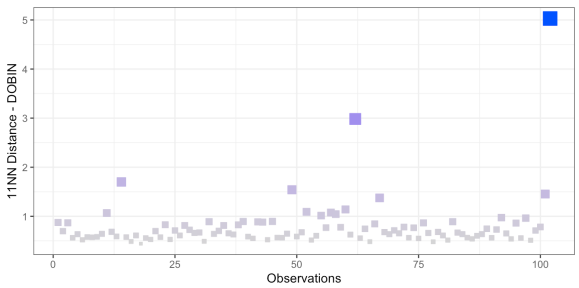
It is on the transformed space (which plays an analogous role to the subspace projection of $\mathbf{X}$ in PCA) that we apply the various outlier and anomaly detection algorithms.

The full details contain a fair number of technical complications; the interested reader is invited to consult the original documentation [18].

Example DOBIN is implemented in `R` *via* the `dobin` package. In the example below, note that the data is **not scaled**.

```
out <- dobin::dobin(rdata[,1:4], frac=0.9, norm=3)

#  arbitrary choice to keep kk not too large
# but respond to dataset size
kk <- min(ceiling(dim(rdata)[1]/10),25)

# dimension reduction
knn_dist <- FNN::knn.dist(out$coords[, 1:3], k = kk)
knn_dist <- knn_dist[ ,kk]
ord <- order(knn_dist, decreasing=TRUE)

knn_dist.dobin <- data.frame(1:(nobs+2),knn_dist)
names(knn_dist.dobin) = c("obs","knn_dobin")
knn_dist.dobin <-
      knn_dist.dobin[order(-knn_dist.dobin$knn_dobin),]
rownames(knn_dist.dobin) <- NULL
head(knn_dist.dobin)

knn_dist.dobin |>
  ggplot(aes(x=obs,y=knn_dobin)) +
  scale_x_continuous(name="Observations") +
  scale_y_continuous(name="11NN Distance - DOBIN") +
  geom_point(aes(fill=knn_dobin, colour=knn_dobin,
                 size=knn_dobin), pch=22) +
  scale_fill_continuous(high = "#0033FF",
                        low = "#CCCCCC") +
  scale_colour_continuous(high = "#0033FF",
                          low = "#CCCCCC") +
  theme_bw() + theme(legend.position = "none")
```

```
obs knn_dobin
102 5.029217
62  2.982912
14  1.700037
49  1.540807
101 1.453970
67  1.375444
```



Look: observation 101 got caught in the DOBIN net!

Algorithm 8: FeatureBagging

Input: dataset D
$j = 1$;
while *stopping criteria are not met* **do**
  Sample an integer r between $p/2$ et $p-1$;
  Randomly select r features (variables) of D in order to create a projected dataset $\tilde{D}_r$ in the corresponding $r-$dimensional sub-space;
  Compute the LOF result for each observation in the projected $\tilde{D}_r$;
  $j = j + 1$;
end
Output: anomaly scores given by the independent ensemble method (average, minimal rank, etc.).

Figure 26.27: Feature bagging algorithm.

26.4.3 Subspace Methods

Subspace methods have been used particularly effectively by analysts for anomaly and outlier detection in high-dimensional datasets [2, 14, 20]; it is often easier to find the sought-after observations by exploring **lower-dimensional subspaces** (rather than the original set) – in that sense, subspace methods are akin to **feature selection** methods, whereas projection methods are closer to **dimension reduction** methods.

There is thus an intrinsic interest in exploring subspaces in their own right [1, 4]. This approach eliminates **additive noise effects** often found in high dimensional spaces and leads to more **robust outliers** (that is, outliers which are identified as such by multiple methods).

The problem is rather difficult to solve effectively and efficiently, since the potential number of subspace projections of high-dimensional data is related **exponentially to the number of features** in the dataset. The **Feature Bagging** algorithm formally uses the LOF algorithm of Section 26.2, but any fast anomaly detection algorithm could also be used instead. The anomaly scores and rankings from each run are aggregated as they are in the **Independent Ensemble** approach (see Section 26.4.4).

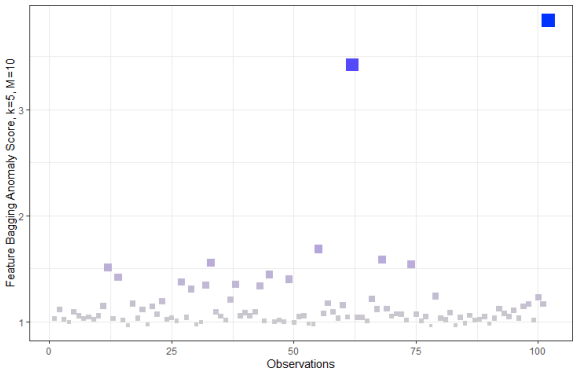
There are other, more sophisticated, subspace anomaly detection methods, including:

- **High-dimensional Outlying Subspaces** (HOS) [39];
- **Subspace Outlier Degree** (SOD) [19];
- **Projected Clustering Ensembles** (OutRank) [29];
- **Local Selection of Subspace Projections** (OUTRES) [27].

Example The feature bagging algorithm is implemented in `R` *via* the `HighDimOut` package (not available on CRAN as of May 2022, but it has been archived . We apply it to our trusty artificial dataset.

```
res.FBOD <- HighDimOut::Func.FBOD(data = rdata[,1:4],
                                  iter=10, k.nn=5)
rdata.FBOD <- data.frame(1:(nobs+2),res.FBOD)
names(rdata.FBOD) = c("obs","FBOD")
rdata.FBOD <- rdata.FBOD[order(-rdata.FBOD$FBOD),]
head(rdata.FBOD)

rdata.FBOD %>%
  ggplot(aes(x=obs,y=FBOD)) +
  scale_x_continuous(name="Observations") +
  scale_y_continuous(name="Feature Bagging Anomaly Score,
                           k=5, M=10") +
  geom_point(aes(fill=FBOD, colour=FBOD, size=FBOD),
                    pch=22) +
  scale_fill_continuous(high = "#0033FF",
                        low = "#CCCCCC") +
  scale_colour_continuous(high = "#0033FF",
                        low = "#CCCCCC") +
  theme_bw() + theme(legend.position = "none")
```

```
obs FBOD
102 3.82
62  3.45
55  1.64
67  1.61
33  1.58
74  1.52
```



26.4.4 Ensemble Methods

In the preceding sections, we have described various anomaly detection algorithms whose relative performance varies with the type of data being considered. As is the case with pretty much of all of data science, it is impossible to come up with an algorithm that outperforms all the others [38].

This is because a particular anomaly detection algorithm may be well-adapted to a dataset and may be successful in detecting abnormal or outlier observations, but it may not work with other datasets whose characteristics do not match the first dataset. The impact of such a mismatch between algorithms can be mitigated by using **ensemble methods**, where the results of several algorithms are considered before making a final decision. Such an approach often provides the best results and thus improves the performance of the base anomaly detection algorithms [26].

We will consider two types of ensemble methods: **sequential ensembles** (boosting) and **independent ensembles**.

Algorithm 6: SequentialEnsemble

```
Inputs: dataset D, base algorithms A_1, ..., A_r
j = 1;
while stopping criteria are not met do
    Select an algorithm A_j based on the results from
     the preceding steps;
    Create a new dataset D_j from D by modifying
     the weight of each observation based on the
     results from the preceding steps;
    Apply A_j to D_j;
    j = j + 1;
end
Output: anomalous observations obtained by
 weighing the results of all previous steps
```

Figure 26.28: Sequential ensemble algorithm.

Sequential Ensembles Sequential ensembles require a given algorithm (or a set of algorithms) to be applied to a dataset in a sequential manner, each time on slightly different dataset derived from the previous step's dataset based on the previous steps' results, and so forth. At each step, the weight associated with each observation is modified according to the preceding results using some "boosting" method.[43] The final result is either some weighted combination of all preceding results, or simply the results output by the last step in the sequence.

43: Such as AdaBoost or XGBoost, for instance.

The formal procedure is provided in Figure 26.28.[44]

44: The details are out-of-scope for this chapter, but they are available in Section 21.5.3.

Independent Ensembles In an independent ensemble, we instead apply different algorithms (or different instances of the same algorithm) to the dataset (or a **resampled dataset**). Choices made at the data and algorithm level are **independent** of the results obtained in previous runs.[45] The results are then combined to obtain more robust outliers.

45: Unlike in a sequential ensemble.

Every **base** anomaly detection algorithm provides an **anomaly score** for each observation in D; observations with higher scores are considered to be more anomalous, observations with lower scores more normal.

The results are then combined using a task-specific method in order to provide a more **robust** classification of anomalous or outlying observations. Many such combination techniques used in practice:

- **majority vote**,
- **average**,
- **minimal rank**, etc.

Let $\alpha_i(\mathbf{p})$ represent the (normalized) **anomaly score** of $\mathbf{p} \in D$, according to algorithm A_i. If $\alpha_i(\mathbf{p}) \approx 0$, it is unlikely that $\mathbf{p}$ is an anomaly according to A_i, whereas if $\alpha_i(\mathbf{p}) \approx 1$, it is quite likely that $\mathbf{p}$ according to A_i.

The **rank** of $\mathbf{p} \in D$ according to A_i, on the other hand, is denoted by $r_i(\mathbf{p})$: the higher the rank (smaller number), the higher the anomaly score and *vice versa*. In a dataset with n observations, the rank varies from 1 to n.[46]

46: Ties are allowed.

Algorithm 7: IndependantEnsemble

Inputs: dataset D, base algorithms $A_1, \ldots, A_r$
$j = 1$;
while *stopping criteria are not met* **do**
  Select an algorithm A_j;
  Create a new dataset D_j from D by (potential) re-sampling, but independently of the preceding steps' results;
  Apply A_j to D_j;
  $j = j + 1$;
end
Output: anomalous observations obtained by combining the results of all previous steps

Figure 26.29: Independent ensemble algorithm.

If the base detection algorithms are $A_1, \ldots, A_m$, the anomaly score and the rank of an observation $\mathbf{p} \in D$ according to the independent ensemble method are, respectively,

$$\alpha(\mathbf{p}) = \frac{1}{m} \sum_{i=1}^{m} \alpha_i(\mathbf{p}) \qquad \text{and} \qquad r(\mathbf{p}) = \min_{1 \le i \le m} \{r_i(\mathbf{p})\}.$$

If $n = m = 3$, for instance, we could end up with

$$\begin{aligned} &\alpha_1(\mathbf{p}_1) = 1.0,\ \alpha_1(\mathbf{p}_2) = 0.9,\ \alpha_1(\mathbf{p}_3) = 0.0; \\ &\alpha_2(\mathbf{p}_1) = 1.0,\ \alpha_2(\mathbf{p}_2) = 0.8,\ \alpha_2(\mathbf{p}_3) = 0.0; \\ &\alpha_3(\mathbf{p}_1) = 0.1,\ \alpha_3(\mathbf{p}_2) = 1.0,\ \alpha_3(\mathbf{p}_3) = 0.0. \end{aligned}$$

Using the mean as the combination techniques, we obtain

$$\alpha(\mathbf{p}_1) = 0.7,\ \alpha(\mathbf{p}_2) = 0.9,\ \alpha(\mathbf{p}_3) = 0.0,$$

whence

$$\mathbf{p}_2 \succeq \mathbf{p}_1 \succeq \mathbf{p}_3,$$

that is, $\mathbf{p}_2$ is more anomalous than $\mathbf{p}_1$, which is more anomalous than $\mathbf{p}_3$.[47]

47: We are using the notation introduced in Section 26.1.2.

Using the minimal rank method, we obtain

$$\begin{aligned} &r_1(\mathbf{p}_1) = 1,\ r_1(\mathbf{p}_2) = 2,\ r_1(\mathbf{p}_3) = 3; \\ &r_2(\mathbf{p}_1) = 1,\ r_2(\mathbf{p}_2) = 2,\ r_2(\mathbf{p}_3) = 3; \\ &r_3(\mathbf{p}_1) = 2,\ r_3(\mathbf{p}_2) = 1,\ r_3(\mathbf{p}_3) = 3, \end{aligned}$$

from which

$$r(\mathbf{p}_1) = r(\mathbf{p}_2) = 1,\ r(\mathbf{p}_3) = 3,$$

and so $\mathbf{p}_1 \succeq \mathbf{p}_3$ and $\mathbf{p}_2 \succeq \mathbf{p}_3$, but $\mathbf{p}_1$, $\mathbf{p}_2$ have the same anomaly levels.

Evidently, the results depend not only on the dataset under consideration and on the base algorithms that are used in the ensemble, but also on **how they are combined**.

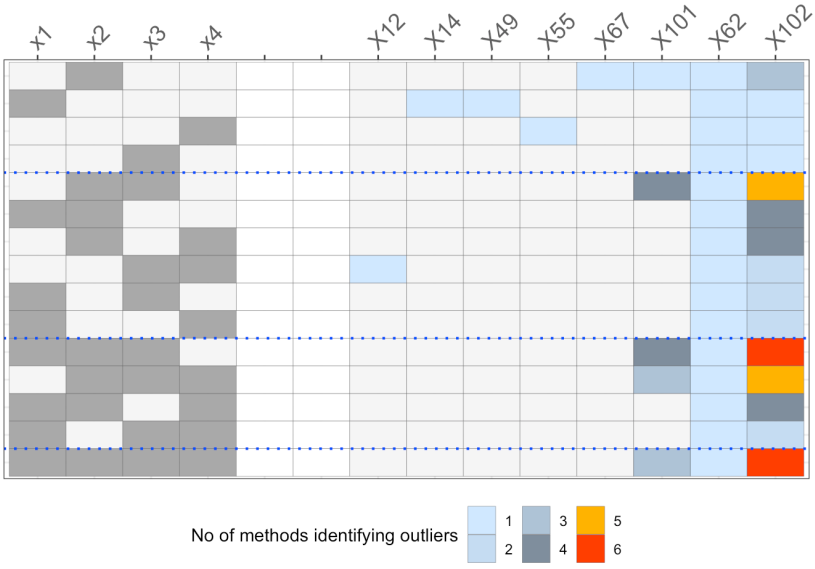
In the context of **HDLSS data**, ensemble methods can sometimes allow the analyst to mitigate some of the effects of the curse of dimensionality by selecting fast base algorithms (which can be run multiple times) and focusing on building robust relative anomaly scores.

Another suggested approach is to use a different sub-collection of the original dataset's features at each step, in in order to **de-correlate** the base detection models (see **feature bagging**, in Section 26.4.3).

Even without combining the results, it may be useful to run multiple algorithms on different subspaces to produce an **Overview of Outliers** (O3), implemented in the `R` package `Outliers03`, by A. Unwin.

```
O3d <- Outliers03::O3prep(rdata[,1:4],
    method=c("HDo", "PCS", "BAC", "adjOut", "DDC", "MCD"))
O3d1 <- Outliers03::O3plotM(O3d)
cx <- data.frame(outlier_method=names(O3d1$nOut),
                 number_of_outliers=O3d1$nOut)
table(cx, row.names=FALSE)
O3d1$gO3
```

```
outlier_method  number_of_outliers
HDo             8
PCS             1
BAC             2
adjOut          2
DDC             1
MCD             2
```



The columns on the left indicate the **subspace variables** (see row colouring). The columns on the right indicate which **observations were identified as outliers** by at least 1 method in at least 1 subspace.[48]

The **colours** depict the number of methods that identify each observation in each subspace as an outlier. For instance, Observation 102 is identified as an outlier by 6 methods in 2 subspaces, 5 methods in 3 subspaces,

48: The available methods are all methods that we have not discussed: `HDoutliers()` from the package `HDoutliers`, `FastPCS()` from the package `FastPCS`, `mvBACON()` from `robustX`, `adjOutlyingness()`, `covMcd()` from `robustbase`, `DectectDeviatingCells()` from `cellWise`.

4 methods in 2 subspaces, 3 methods in 1 subspace, 2 methods in 4 subspaces, and 1 method in 3 subspaces – it is clearly the **most anomalous** observation in the dataset. Observations 62 and 101 are also commonly identified as outliers.

Are the results aligned with those we have obtained throughout the chapters?

Ensemble approaches allow analysts to take a big picture view of the anomaly landscape, but it should be recalled that anomaly detection and outlier analysis is still very active as an area of research, with numerous challenges. The *No Free Lunch* Theorem suggests that, importantly, **there is no magic method**: all methods have strengths and limitations, and the results depend heavily on the data.

26.5 Exercises

1. Use other metrics and parameter values to find distance-based anomalies, LOF outliers, DBSCAN/HDBSCAN/OPTICS outliers, and Isolation Forest outliers in the artificial dataset.
2. Consider the datasets:
   - `GlobalCitiesPBI.csv`
   - `2016collisionsfinal.csv`
   - `polls_us_election_2016.csv`
   - `HR_2016_Census_simple.xlsx`
   - `UniversalBank.csv`
   - `algae_blooms.csv`

   a) Find distance-based anomalies in the datasets.
   b) Find density-based anomalies in the datasets.
   c) Find categorical anomalies in the datasets that have categorical features.
   d) Find projection-based anomalies in the datasets.
   e) Find subspace-based anomalies in the datasets
   f) Find ensemble-based anomalies in the datasets.
3. Conduct an analysis of anomalous observations in the 2011 Gapminder data (as described in Chapters 20, 21, 22, and 23).
4. Consider the dataset `flights1_2019_1.csv`.

   a) Explore and visualize the dataset.
   b) Do any observations appear to be anomalous or outlying? Justify your answer.
   c) If necessary, reduce the dimension or transform the dataset prior to analysis.
   d) Using at least 4 anomaly detection algorithms, identify anomalous observations in the dataset.
   e) Can you validate the results?

Chapter References

[1] C.C. Aggarwal. *Outlier Analysis*. Springer International Publishing, 2016.

[2] C.C. Aggarwal and S. Sathe. *Outlier Ensembles: An Introduction*. Springer International Publishing, 2017.

[3] C.C. Aggarwal and P.S. Yu. 'Outlier detection for high dimensional data'. In: *SIGMOD Rec.* 30.2 (2001), pp. 37–46. DOI: http://doi.acm.org/10.1145/376284.375668.

[4] A. Archimbaud. 'Détection non-supervisée d'observations atypiques en contrôle de qualité : un survol'. In: *Journal de la Société Française de Statistique* 159.3 (2018), pp. 1–39.

[5] L. Arora. *An Awesome Tutorial to Learn Outlier Detection in Python Using PyOD* . Analytics Vidhya. 2019.

[6] D. Baron. *Outlier Detection* . XXX Winter School of Astrophysics on Big Data in Astronomy, GitHub repository. 2018.

[7] M.M. Breunig et al. 'LOF: Identifying Density-Based Local Outliers'. In: *SIGMOD Rec.* 29.2 (2000), pp. 93–104.

[8] R.J.G.B. Campello, D. Moulavi, and J. Sander. 'Density-Based Clustering Based on Hierarchical Density Estimates'. In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Jian Pei et al. Springer Berlin Heidelberg, 2013, pp. 160–172.

[9] C. Chen and L.-M. Liu. 'Joint Estimation of Model Parameters and Outlier Effects in Time Series'. In: *Journal of the American Statistical Association* 88 (Mar. 1993). DOI: 10.2307/2290724.

[10] M. Ester et al. 'A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise'. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. AAAI Press, 1996, pp. 226–231.

[11] P. Gogoi et al. 'A Survey of Outlier Detection Methods in Network Anomaly Identification.' In: *Comput. J.* 54.4 (2011), pp. 570–588.

[12] S. Hariri, M. Carrasco Kind, and R. J. Brunner. 'Extended Isolation Forest'. In: *IEEE Transactions on Knowledge and Data Engineering* (2019).

[13] T. Hastie. *Leukemia dataset* .

[14] Z. He, S. Deng, and X. Xu. 'A Unified Subspace Outlier Ensemble Framework for Outlier Detection'. In: *Advances in Web-Age Information Management*. Ed. by Wenfei Fan, Zhaohui Wu, and Jun Yang. Springer Berlin Heidelberg, 2005, pp. 632–637.

[15] Z. He et al. 'A Fast Greedy Algorithm for Outlier Mining'. In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Wee-Keong Ng et al. Springer, 2006, pp. 567–576.

[16] *Height Percentile Calculator, by Age and Country* .

[17] V. Hodge and J. Austin. 'A survey of outlier detection methodologies'. In: *Artif. Intell. Rev.* 22.2 (2004), pp. 85–126.

[18] S. Kandanaarachchi and R. Hyndman. *Dimension reduction for outlier detection using DOBIN*. Sept. 2019. DOI: 10.13140/RG.2.2.15437.18403.

[19] H.-P. Kriegel et al. 'Outlier Detection in Axis-Parallel Subspaces of High Dimensional Data'. In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Thanaruk Theeramunkong et al. Springer Berlin Heidelberg, 2009, pp. 831–838.

[20] A. Lazarević and V. Kumar. 'Feature Bagging for Outlier Detection'. In: *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. KDD '05. Association for Computing Machinery, 2005, pp. 157–166. DOI: 10.1145/1081870.1081891.

[21] A. Lazarević, J. Srivastava, and V. Kumar. *Data Mining for Analysis of Rare Events: A Case Study in Security, Financial and Medical Applications* . Tutorial. 2004.

[22] T. Le et al. 'A Hybrid Approach Using Oversampling Technique and Cost-Sensitive Learning for Bankruptcy Prediction'. In: *Complexity* (2019). DOI: 10.1155/2019/8460934.

[23] F.T. Liu, K.M. Ting, and Z.-H. Zhou. 'Isolation Forest'. In: *Proceedings of the Eighth IEEE International Conference on Data Mining*. Ed. by Fosca Giannotti et al. IEEE International Conference on Data Mining 2008, ICDM 2008. IEEE, 2008, pp. 413–422.

[24] O. Maimon and L. Rokach, eds. *Data Mining and Knowledge Discovery Handbook*. 2nd ed. New York: Springer, 2010.

[25] L. McInnes, J. Healy, and S. Astels. *How HDBSCAN Works*. 2016.

[26] K.G. Mehrotra, C.K. Mohan, and H.M. Huang. *Anomaly Detection Principles and Algorithms*. Springer, 2017.

[27] E. Müller, M. Schiffer, and T. Seidl. 'Adaptive Outlierness for Subspace Outlier Ranking'. In: *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*. CIKM '10. Toronto, ON, Canada: Association for Computing Machinery, 2010, pp. 1629–1632.

[28] E. Müller, M. Schiffer, and T. Seidl. 'Statistical Selection of Relevant Subspace Projections for Outlier Ranking'. In: *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*. ICDE '11. IEEE Computer Society, 2011, pp. 434–445. DOI: 10.1109/ICDE.2011.5767916.

[29] E. Müller et al. 'Outlier Ranking via Subspace Analysis in Multiple Views of the Data'. In: *2012 IEEE 12th International Conference on Data Mining*. 2012, pp. 529–538. DOI: 10.1109/ICDM.2012.112.

[30] E. Müller et al. 'OutRank: Ranking Outliers in High-Dimensional Data'. In: *2008 IEEE 24th International Conference on Data Engineering Workshop*. 2008, pp. 600–603. DOI: 10.1109/ICDEW.2008.4498387.

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

[32] *Outlier Detection: a Survey*. Technical Report TR 07-017. 2007.

[33] O. Soufan et al. 'Mining Chemical Activity Status from High-Throughput Screening Assays'. In: *PloS one* 10.12 (2015). DOI: 10.1371/journal.pone.0144426.

[34] P.D. Talagala, R.J. Hyndman, and K. Smith-Miles. *Anomaly Detection in High Dimensional Data*. 2019. DOI: 10.48550/ARXIV.1908.04000.

[35] J. Tang et al. 'Capabilities of Outlier Detection Schemes in Large Datasets, Framework and Methodologies'. In: *Knowl. Inf. Syst.* 11.1 (Jan. 2007), pp. 45–84.

[36] D. Whitenack. *Machine Learning With Go*. Packt Publishing, 2017.

[37] Wikipedia. 'Matthews Correlation Coefficient'. In: ().

[38] D.H. Wolpert and W.G. Macready. 'No free lunch theorems for optimization'. In: *IEEE Transactions on Evolutionary Computation* (1997).

[39] J. Zhang et al. 'Hos-Miner: A System for Detecting Outlyting Subspaces of High-Dimensional Data'. In: *Proceedings of the Thirtieth International Conference on Very Large Data Bases*. VLDB '04. Toronto, Canada: VLDB Endowment, 2004, pp. 1265–1268.

Text Analysis and Text Mining 27

by **Patrick Boily**, with contributions from **Andrew Macfie**

In this chapter, we introduce the basic notions of **text mining** so that we can learn how to extract insight from text data. We also present some elementary applications (drawn from the world of machine learning), including **sentiment analysis**, setting the stage for a more sophisticated treatment of **natural language processing** and **large language models** (see Chapters 31 and 32).

27.1 Introduction

We start our foray into text analysis by discussing a use case for text mining which made the news early in 2017.

27.1.1 Case Study: BOTUS

In 2013, the BBC reported on various ways in which social media giant Twitter was changing the world, detailing specific instances in the fields of business, politics, journalism, sports, entertainment, activism, arts, and law [27].

It is not always clear what influence Twitter users have, if any, on world events or business and cultural trends; it was once thought (perhaps without appropriate evidence) that entertainers, athletes, and celebrities, that is to say, users with extremely high followers to following ratios, wielded more "influence" on the platform than world leaders [4]. Certainly, such users continue to be among the most popular – as of September 13, 2017, Twitter's 40 most-followed accounts tend to belong to entertainers, celebrities, and athletes, with a few exceptions [11].

One account has recently bridged the gap between celebrity and politics in an explosive manner: `@realDonaldTrump`, which belongs to the 45th President of the United States of America, has maintained a very strong presence on Twitter. As of September 13, 2017, the account had 38,205,766 followers, and it was the 26th most-followed account on the planet, producing 35,755 tweets since it was activated in March 2009, and roughly 6 tweets a day in August of 2017 [11].

Titles BOTUS [15], Trump & Dump Bot [42]

Authors Tradeworx (BOTUS), T3 (Trump & Dump)

Date 2017

Sponsor NPR's podcast *Planet Money* (BOTUS)

Methods sentiment analysis, social media monitoring, AI, real-time analysis, simulations

Objective There is some evidence to suggest that tweets from the 45th POTUS may have an effect on the stock market [21]. Can sentiment analysis and AI be used to take real-time advantage of the tweets' unpredictable nature? Let's take a look at bots built for that purpose by NPR's *Planet Money* and by T3 (an Auston advertising agency).

Methodology Tradeworx followed these steps:

1. *Data collection*: tweets from `@realDonaldTrump` are collected for analysis.
2. *Sentiment analysis of tweets*: each tweet is given a sentiment score on the positive/negative axis.
3. *Validation*: the sentiment analysis scoring must be validated by observers: are human-identified positive or negative tweets correctly identified by BOTUS?
4. *Identification of the company in a tweet*: is the tweet even about a company? If so, which one?
5. *Determining the trading universe*: are there companies that should be excluded from the bot's trading algorithms?
6. *Classifying tweets as "applicable" or "unapplicable"*: is a tweet's sentiment strong enough for BOTUS to engage the trading strategy?
7. *Determining a trading strategy*: how soon after a flagged tweet does BOTUS buy a company's stock, and how long does it hold it for?
8. *Testing the trading strategy on past data*: how would BOTUS have fared from the U.S. Presidential Election to April 2017? What are BOTUS' limitations?

T3's Trump and Dump uses a similar process (see Figure 27.1).

Data The data consists of:

- tweets by `@realDonaldTrump` (from around Election Day 2016 through the end of March 2017 for BOTUS; no details are given for T3) (see Figure 27.2 for sample);
- a database of publicly traded companies, such as can be found at [17, 18, 14], although which of these were used, if any, is not specified (no explicit mention is made for BOTUS), and
- stock market data for real-time pricing (Google Finance for T3) and backcasting simulation (for BOTUS, source unknown).

It is not publicly known whether the 2 bots are upgrading their algorithms by including new data as time passes.

Figure 27.1: T3's Trump and Dump process [42].

Strengths and Limitations of Algorithms and Procedure

- In sentiment analysis, an algorithm analyzes documents in an attempt to identify the attitude they express or the emotional response they seek. It presents numerous challenges, mostly related to the richness and flexibility of human languages and their syntax variations, the context-dependent meaning of words and lexemes, the use of sarcasm and figures of speech, and the lack of perfect inter-rater reliability among humans [35]. As it happens, `@realDonaldTrump` is not much of an ironic tweeter – "sad" and "great" are usually meant in their most general sense. This greatly simplifies the analysis.
- The bots have to learn to recognize whether a tweet is directed at a publicly traded company or not. In certain cases, the ambiguity can be resolved relatively easily with an appropriate training set (Apple the company vs. apple the food-item, say), but no easy solutions were found in others (Tiffany the company vs. Tiffany the daughter, for example). Rather than have humans step in and instruct BOTUS when it faces uncertainty (which would go against the purpose of the exercise), a decision was made to exclude these cases from the trading universe. The T3 documentation does not describe such details.

Figure 27.2: Examples of `@realDonaldTrump` tweets involving Delta, Toyota Motor, L.L.Bean, Ford, and Boeing.

- Once the bot knows how to rate `@realDonaldTrump`'s tweets and to identify when he tweets about publicly-traded companies, the next question is to determine what the trading strategy should be. If the tweet's sentiment is negative enough T3 shorts the company's stock.[1] Of course, this requires first purchasing the stock (so that it can be shorted). Planet Money's decision was similar: buy once the tweet is flagged, and sell right away... but what does "right away" mean in this context? There is a risk involved: if the stock goes back up before BOTUS has had a chance to purchase the low-priced stock, it will lose money. To answer that question, Tradeworx simulated the stock market over the last few months, introducing the tweets, and trying out different trading strategies. It turns out that, in this specific analysis, "right away" can be taken to be 30 minutes after the tweet.

1: It sells the stock when the price is high, that is, *before* the tweet has had the chance to bring the stock down, and it repurchases it once the price has been lowered by the tweet, but before the stock has had the chance to recover.

Results, Evaluation and Validation For a trading bot, the validation is in the pudding, as they say – do they make money? T3's president says that their bot is profitable (they donate the proceeds to the ASPCA) [42]: for instance, they netted a return of 4.47% on `@realDonaldTrump`'s Delta tweet (see Figure 27.2); however, he declined to provide specific numbers (and made vague statements about providing monthly reports, which I have not been able to locate) [31].

The BOTUS process was more transparent, and we can point to Planet Money's transcript for a discussion on sentiment analysis validation (comparing BOTUS's sentiment rankings with those provided by human observers, or running multiple simulations to determine the best trading scenario) [15] – but it suffers from a serious impediment: as of roughly 4 months after going online, **it still had not made a single trade** [13]!

The reasons are varied (see Figures 27.3 and 27.4), but the most important setback was that `@realDonaldTrump` had not made a single valid tweet about a public company whose stock BOTUS could trade during the stock market business hours. Undeterred, Planet Money relaxed its trading strategy: if `@realDonaldTrump` tweets during off-hours, BOTUS will short the stock at the market's opening bell.

This is a risky approach, and so far it has not proven very effective: a single trade of Facebook's trade, on August 23rd, which resulted in a loss of 0.30$ (see Figure 27.4).

Figure 27.3: BOTUS reporting on its trades (part 1).

Take-Aways As a text analysis and scenario analysis project, both BOTUS and Trump & Dump are successful – they present well-executed sentiment analyses, and a simulation process that finds an optimal trading strategy. As predictive tools, they are sub-par (as far as we can tell), but for reasons that (seem to) have little to do with data analysis *per se*.

Figure 27.4: BOTUS reporting on its trades (part 2).

Unfortunately, this is not an atypical feature of descriptive data analysis: we can explain what has happened (or what is happening), but the modeling assumptions are not always applicable to the predictive domain.

27.1.2 Text Analysis

It has been said that "the only valid model of the universe might just be the universe itself" [author unknown]. With that maxim in mind, it would appear that there is simply no substitute – in order to get meaning out of documents, we first need to read them in their entirety.

This interpretation is overly simplistic, however. Consider the works of William Shakespeare, for instance, to whom 38 plays (or so) and over 150 sonnets and poems written in the late 1600s and early 1700s have been attributed [48]. It is fairly straightforward to have a go at *Macbeth*, say; one only needs to pick up a printed or digital copy, or sit through any of the numerous stagings of the Scottish play, and voilà! – instant meaning and themes: ambition, lust for power, appearances vs. reality, temptation, and guilt haunting evildoers [6].

Of course, current readers might find Early Modern English verses difficult to follow without annotation, and those themes might only

reveal themselves upon repeated readings or viewings. Lovers of English verse might fully enjoy this arduous process, but non-native speakers might wonder if data analysis methods could provide a complete (or near enough) Shakespeare experience without having to go through *The Complete Works of Shakespeare*, or even *The Complete Works of William Shakespeare (Abridged)* [7, 28]? Is there some "essential" Shakespeare-ness that lurks in his plays and sonnets? Common threads, common humour, common structure, common themes?

These questions (and others, such as **authorship questions** [10, 19, 49, 12]) may only be of interest to scholars, but there is a more compelling reason to study automated text analysis, if only as a first pass – in the age of "fake news" [30, 22, 29, 2], social media, and Amazon reviews, when the tweets of high-profile politicians have a definite and measurable influence on the stock market [13, 15, 16, 5, 33, 34] or when live analysis of panic conversations can drive automated emergency responses [38], we simply produce too much text data for any group of individuals to analyze and understand without technological assistance.[2]

2: In other words, the genie is out of the bottle – what can we do to make sure we understand what it's *really* saying?

27.1.3 Text Mining vs. Natural Language Processing

Text mining is the collection of processes by which we can extract useful insights from text. Inherent in this definition is the idea of **automated data reduction**: useful insights (whether in the form of summaries, sentiment analyses, or word counts) ought to be "smaller" and "more organized" (from a data point of view) than the original text.

For short texts, however, the benefits of text analysis may not always be evident. Consider, for instance, the following excerpt from a lawn mowing instruction manual:

> Before starting your mower inspect it carefully to ensure that there are no loose parts and that it is in good working order.

This is a fairly **concise** and **structured** way to convey a message. It could be further shortened and organized, perhaps, but it's not clear that one would gain much from the process. In more complex case, the process is less obvious; we discussed data reduction in a more general context previously and encourage readers of this chapter to first take a look at Section 23.1 (*Data Reduction for Insight*).

Ted Kwartler suggests the following **text mining workflow** [26]:

1. problem definition and goals;
2. identify text to be collected;
3. text organization;
4. feature extraction;
5. analysis, and
6. reach an insight, conclusion, or output.

In this chapter, we will further take the position that text mining is the application of **data science and machine learning tasks to text documents**, such as:

- **supervised learning** (classification and class probability estimation, value estimation);

Figure 27.5: A poutine (on the left); an abomination in the eyes of all right-thinking sentient beings (on the right).

- **unsupervised learning** (association rules and hypothesis discovery, similarity matching, clustering);
- **semi-supervised learning** (profiling, link prediction, data reduction, etc.), and
- **visualization and representation**.

Typical applications include **authorship questions** (classification), **sentiment analysis** (value estimation), **taxonomy creation** and **topic modeling** (clustering), **text description**, and **text visualization**.[3]

We will explore the data preparation process and simple text mining models in Sections 27.2 and 27.3, respectively.

Natural language processing (NLP), in contrast, has a long history of lofty goals, which more or less boil down to developing machines that **react "appropriately" while interacting with (natural) human languages**.[4] The focus of NLP tasks tends towards **"understanding" languages**; with common tasks including:

- **syntax** (lemmatization, part-of-speech tagging, parsing, terminology extraction, sentence boundary disambiguation, stemming, word segmentation, etc.);
- **semantics** (machine translation, language generation, named entity recognition, optical character recognition, questions and answers, sentiment analysis, textual entailment, topic segmentation, word sense disambiguation, etc.);
- **discourse** (coreference resolution, discourse analysis, summarization, etc.), and
- **speech** (recognition, segmentation, text-to-speech, etc.). [47]

Most natural human languages rules are **dynamic**, and usage may change drastically in space and time – a *poutine* is not the same dish in New Brunswick as it is in Québec, for instance (see Figure 27.5). For another example, consider the meaning of the word *awful*, which drifted from

> "commanding profound respect or reverential fear"

to

> "frightful, very ugly, monstrous"

from 1000 AD onward.

Other issues arise from dialect variations and individual-specific speech patterns, either due to **linguistic drift**, **influence from other languages**, **sarcasm**, **idioms**, **figures of speech**, and so forth. The intended meaning is often clear to experienced human speakers based on the specific context, but it is believed that natural language understanding is **AI-hard** – a

3: In order to take full advantage of the underlying data science machinery, documents may first need to be mapped into numerical or categorical features, via **kernel transformations**.

4: Think of ChatGPT, as a recent example.

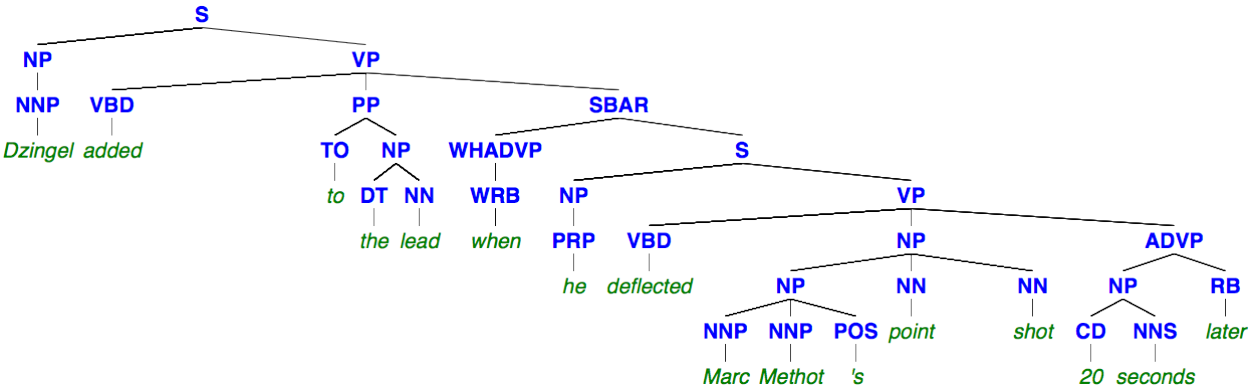


Figure 27.6: Syntactic parsing of a sentence using the Stanford parser [41].

complete resolution of the issues would require the ability to make computers as "smart" as humans [20], in ways that we haven't seen from our machines yet.[5] Thankfully, we rarely need a resolution at the most general level; in many areas, the current state of the art produces results which are acceptable to a large class of users.

5: Scratching the surface of large language models shows how far we remain, in spite of recent progresses.

Since the 1990s, the NLP community has adopted a machine learning paradigm, which has provided advantages over the classical hard-coded hand-produced rules. Statistical machine translation, for instance, can take advantage of domain constraints and formal language habits to reduce the space of outputs and produce accurate translations of technical documents [45]. We will take a more detailed look at NLP concepts and tasks in Chapter 32.

The distinction between text mining and NLP may seem spurious; most researchers and practitioners do not get bogged down in such details. When we focus on the **data science** side of the equation, we'll refer to text analysis as **text mining**; when we focus on **language analysis and understanding**, we'll refer to it as **natural language processing**.

27.2 Basics of Text Analysis

Let's take a look at two schools of thought regarding text mining: semantic parsing and bag-of-words mining.

Semantic Parsing In this view of text mining, word order and word type play a crucial role. The idea is to use a large number of hand-parsed sentences to train a model that outputs the most likely grammatical analysis of a sentence. Words are tagged along a tree structure, and may have multiple features. This information can then be used to extract insights about the sentence or document.

For instance, consider the sentence

> (S1) Dzingel added to the lead when he deflected Marc Methot's point shot 20 seconds later [37],

a syntactic parsing of which is shown in Figure 27.6.[6]

6: The output of the Stanford parser [41].

Another display is provided below:

```
(ROOT
 (S
  (NP (NNP Dzingel))
  (VP (VBD added)
    (PP (TO to)
      (NP (DT the) (NN lead)))
    (SBAR
      (WHADVP (WRB when))
      (S
        (NP (PRP he))
        (VP (VBD deflected)
          (NP
            (NP (NNP Marc) (NNP Methot) (POS 's))
            (NN point) (NN shot))
          (ADVP
            (NP (CD 20) (NNS seconds))
            (RB later))))))))
```

From the tree diagram, a human observer can clearly see that "Marc Methot" is correctly parsed as a *noun phrase* (NP), that the "'s" is correctly identified as a *possessive marker* (POS), and that "Marc Methot's point shot" is correctly shown as a NP (built from 2 *singular proper nouns*, NNP), but the parser fails to recognize "point shot" as an NP.[7]

7: The two displays are, of course, equivalent. A computer program can be used to easily go from one to the other; a human with the right experience would find both as insightful. But it's certainly easier for a neophyte to comprehend the tree diagram. Why is that? Is it simply because we are a visual species? Or because most of us have parsed sentence fragments in our native languages as youths?

In another parsing (using the Enju parser, see Figure 27.7), "Marc Methot's point shot 20 seconds later" is tagged as a *simple declarative clause* (S), but "Marc Methot's point" and "shot 20 seconds later" are wrongly identified as a NP and a *verb phrase* (VP), respectively, underscoring the importance of parsing to our understanding of a sentence.

The **part-of-speech tagging** for the sentence is shown in the table below:

Word	Tag	Word	Tag
Dzingel	NNP	Marc	NNP
added	VBD	Methot	NNP
to	TO	's	POS
the	DT	point	NN
lead	NN	shot	NN
when	WRB	20	CD
he	PRP	seconds	NNS
deflected	VBD	later	RB

The meaning of common tags are provided in Tables 27.1 and 27.2. Notice how relational insight between the parts-of-speech has gotten lost (or is not displayed, at the very least).

The Stanford parser provides a list of **universal dependencies**:

- nsubj(added-2, Dzingel-1)
- root(ROOT-0, added-2)
- case(lead-5, to-3)
- det(lead-5, the-4)
- nmod(added-2, lead-5)

- advmod(deflected-8, when-6)
- nsubj(deflected-8, he-7)
- advcl(added-2, deflected-8)
- compound(Methot-10, Marc-9)
- nmod:poss(shot-13, Methot-10)
- case(Methot-10, 's-11)
- compound(shot-13, point-12)
- dobj(deflected-8, shot-13)
- nummod(seconds-15, 20-14)
- nmod:npmod(later-16, seconds-15)
- advmod(deflected-8, later-16)

For instance, "he" (the 7th token in the sentence) is he *nominal subject* (nsubj) of "deflected" (the 8th token), "point shot" is recognized as a *compound*, and "shot" (the 13th token) is the *direct object* (dobj, the second most core argument of a verb after the subject) of "deflected" (the 8th token). A list of codes and meanings for UD (v2) can be found in Tables 27.3 and 27.4, on pp. 1718-1719.

Bag of Words In this view of text mining, only the words are important – it is frequency (and relative frequency) that wins the day. In semantic parsing, the words have attributes depending on their position and role in the document's sentences; in bag of words analysis, **the words themselves are attributes of the document**. Our sentence S1 is simply a collection of words, arranged here alphabetically:

> 's, 20, added, deflected, Dzingel, he, later, lead, Marc, Methot, point, seconds, shot, the, to, when.

The fact that "point shot" is a noun phrase is not significant, but the fact that "point" and "shot" appear in the list is significant – it is the **relative frequencies** of the terms that provide information about the document or collection of documents (such as intent and themes).

In the rest of the section, we will take a look at the fundamental concepts underlying text preparation.[8] Concrete illustrations of these notions are provided in Section 27.4.

8: Some of the topics will be revisited in Chapter 32.

27.2.1 Text Collection

Nowadays, text data is typically collected from the Web, either through **web scraping** or with the help of a specialized **application programming interface** (API), as we discussed in Chapter 16. **Manual collection** is another option (although strongly discouraged when faced with more than a few dozen documents to collect).

Optical character readers (OCR) can also digitize scanned images and the technology has improved tremendously over the last 20 years; **manual entry of non-digital text data** can be used as a last resort, but it is tedious and likely to introduce infelicities and transcription errors.

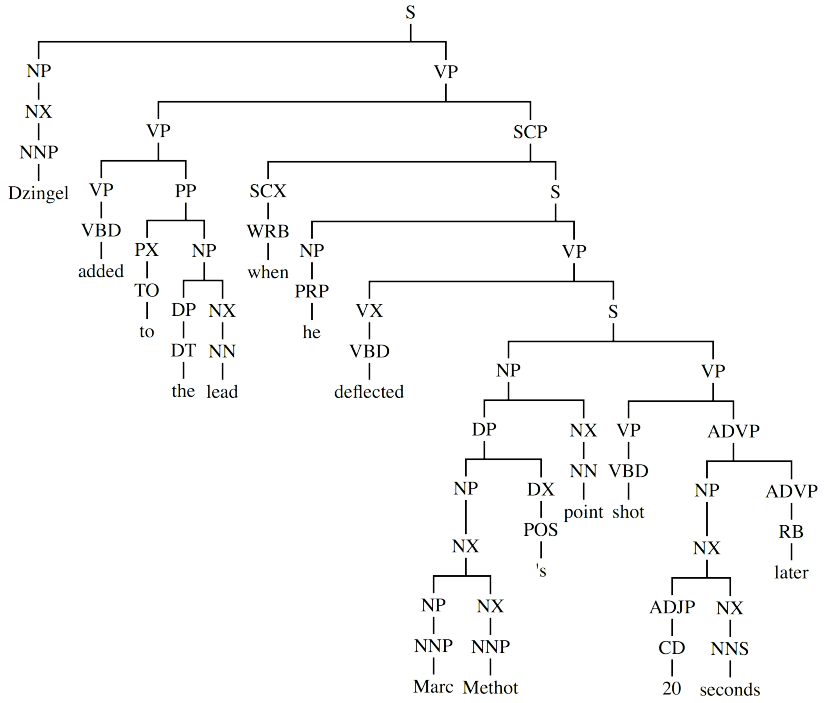


Figure 27.7: Abridged syntactic parsing of a sentence using the Enju English parser [32].

27.2.2 Text Representation

No matter where data comes from and what analyses we hope to run on it, the crucial first step requires extraction, formatting, and storage to a data structure with appropriate numerical properties [39]:

- a **string** or vector of characters, with language-specific encoding;
- a collection of text documents (with meta information) called a **corpus** ('permanent' when stored on disk; 'volatile' when held in RAM);
- a **document-term matrix** (DTM) – or the transposed **term-document matrix** (TDM) – where the rows are the documents, the columns are the terms (see Figure 27.8), and the entries represent some text statistic;
- a **tidy text dataset** containing one **token** (single word, n-gram, sentence, paragraph) per row.

The DTM/TDM representations are **essential** to any statistical analysis of text data – it is on these entities that machine learning algorithms are unleashed.

27.2.3 Text Processing

As with every form of data, text data requires extensive **cleaning** and **processing**. Cleaning text data is, to put it mildly, even less pleasant a process than cleaning numeric or categorical data. There are challenges due to the nature of the data: how would one go about finding anomalies in the text? Outliers? Is the concept even definable for text data?

part-of-speech tagset

CC	Coordinating conj.	TO	infinitival *to*
CD	Cardinal number	UH	Interjection
DT	Determiner	VB	Verb, base form
EX	Existential there	VBD	Verb, past tense
FW	Foreign word	VBG	Verb, gerund/present pple
IN	Preposition	VBN	Verb, past participle
JJ	Adjective	VBP	Verb, non-3rd ps. sg. present
JJR	Adjective, comparative	VBZ	Verb, 3rd ps. sg. present
JJS	Adjective, superlative	WDT	Wh-determiner
LS	List item marker	WP	Wh-pronoun
MD	Modal	WP$	Possessive *wh*-pronoun
NN	Noun, singular or mass	WRB	Wh-adverb
NNS	Noun, plural	#	Pound sign
NNP	Proper noun, singular	$	Dollar sign
NNPS	Proper noun, plural	.	Sentence-final punctuation
PDT	Predeterminer	,	Comma
POS	Possessive ending	:	Colon, semi-colon
PRP	Personal pronoun	(	Left bracket character
PPS	Possessive pronoun	)	Right bracket character
RB	Adverb	"	Straight double quote
RBR	Adverb, comparative	‘	Left open single quote
RBS	Adverb, superlative	“	Left open double quote
RP	Particle	’	Right close single quote
SYM	Symbol	”	Right close double quote

syntactic tagset

ADJP	Adjective phrase
ADVP	Adverb phrase
NP	Noun phrase
PP	Prepositional phrase
S	Simple declarative clause
SBAR	Subordinate clause
SBARQ	Direct question introduced by *wh*-element
SINV	Declarative sentence with subject-aux inversion
SQ	Yes/no questions and subconstituent of SBARQ excluding *wh*-element
VP	Verb phrase
WHADVP	Wh-adverb phrase
WHNP	Wh-noun phrase
WHPP	Wh-prepositional phrase
X	Constituent of unknown or uncertain category
*	“Understood” subject of infinitive or imperative
0	Zero variant of *that* in subordinate clauses
T	Trace of wh-Constituent

Table 27.1: Penn treebank tagset (part 1) [43].

Character encoding may also produces surprises: a text (or a part of text) that looks completely normal to the human eye may be unreadable to a computer because it was expecting a different encoding system. There are probabilistic ways to detect a document's encoding, and ways to coerce a specified encoding – if you are working with text data and your code balks at doing something it should be able to do and none of the usual fixes apply, look into the encoding situation.

Another issue is that legitimate **spelling mistakes** and **typographical errors** are hard to catch in large documents (even with spell-checkers), to say nothing of:

- **accent representation** (*ya new cah's wicked pissa!*);
- **neologisms** and **portemanteaus** (*ruthfull*; *can't you tell that I'm planning prevenge?*);
- **poor translations** or **foreign words** (*business goose*; *llongyfarchiadau*);
- **puns** and **play-on-words** (*they were jung and easily freudened!*);
- **specialized vocabulary** (*clopen*; *poset*);
- **fictional names** and **places** (*Qo'noS*; *Kilgore Trout*);
- **slang** and **curses** (*skengfire*; *#$&#!*);
- **mark-up** and **tags** (*<b>*; *\includegraphics*);
- **uninformative text information** (page number; ISBN blurb), etc.

functional tagset

Text Categories	
-HLN	headlines and datelines
-LST	list markers
-TTL	titles
Grammatical Functions	
-CLF	true clefts
-NOM	non NPs that function as NPs
-ADV	clausal and NP adverbials
-LGS	logical subjects in passives
-PRD	non VP predicates
-SBJ	surface subject
-TPC	topicalized and fronted constituents
-CLR	closely related - see text
Semantic Roles	
-VOC	vocatives
-DIR	direction & trajectory
-LOC	location
-MNR	manner
-PRP	purpose and reason
-TMP	temporal phrases

pseudo-attachments tagset

ICH	Interpret Constituent Here
PPA	Permanent Predictable Ambiguity
RNR	Right Node Raising
EXP	Expletive

disfluency annotations

Utterances	
/	end of complete utterance
-/	end of incomplete utterance
Non-sentence elements	
F	fillers (*uh*, *um*, *huh*, *oh*, etc.)
E	explicit editing term (*I mean*, *sorry*, etc.)
D	discourse marker (*you know*, *well*, etc.)
C	coordinating conjunction (*and*, *and then*, *but*, etc.)
A	aside
Restarts	
[*RM* + *RR*]	restart with repair (see text)
[*RM*+]	restart without repair

Table 27.2: Penn treebank tagset (part 2) [43].

	Nominals	Clauses	Modifier words	Function Words
Core arguments	nsubj obj iobj	csubj ccomp xcomp		
Non-core dependents	obl vocative expl dislocated	advcl	advmod discourse	aux cop mark
Nominal dependents	nmod appos nummod	acl	amod	det clf case
Coordination	**MWE**	**Loose**	**Special**	**Other**
conj cc	fixed flat compound	list parataxis	orphan goeswith reparandum	punct root dep

Table 27.3: The 37 universal syntactic relations used in Universal Dependencies v2. The upper part of the table follows the main organizing principles of the UD taxonomy such that rows correspond to functional categories in relation to the head (core arguments of clausal predicates, non-core dependents of clausal predicates, and dependents of nominals) while columns correspond to structural categories of the dependent (nominals, clauses, modifier words, function words). The lower part of the table lists relations that are not dependency relations in the narrow sense [44].

Code	Relation
acl	clausal modifier of noun (adjectival clause)
advcl	adverbial clause modifier
advmod	adverbial modifier
amod	adjectival modifier
appos	appositional modifier
aux	auxiliary
case	case marking
cc	coordinating conjunction
ccomp	clausal complement
clf	classifier
compound	compound
conj	conjunct
cop	copula
csubj	clausal subject
dep	unspecified dependency
det	determiner
discourse	discourse element
dislocated	dislocated elements
expl	expletive
fixed	fixed multiword expression
flat	flat multiword expression
goeswith	goes with
iobj	indirect object
list	list
mark	marker
nmod	nominal modifier
nsubj	nominal subject
nummod	numeric modifier
obj	object
obl	oblique nominal
orphan	orphan
parataxis	parataxis
punct	punctuation
reparandum	overridden disfluency
root	root
vocative	vocative
xcomp	open clausal complement

Table 27.4: Universal dependency relations, alphabetical listing [44].

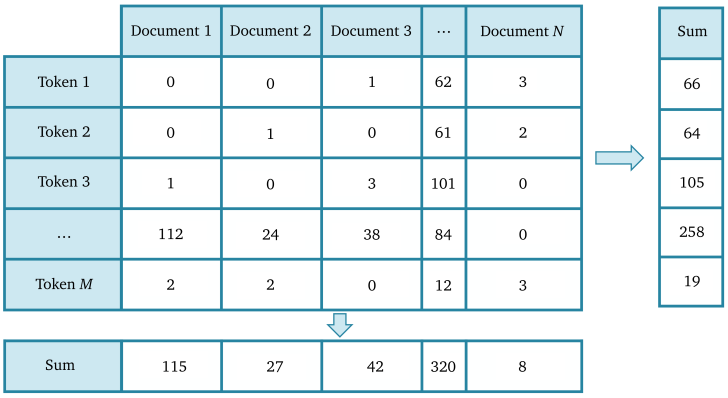
The process can be simplified to some extent with the help of **regular expressions** and **text pre-processing functions** (see Section 27.4):

> Specific pre-processing steps will vary based on the project. For example, the words used in tweets are vastly different than those used in legal documents, so **the cleaning process can also be quite different** [26].

We shall illustrate the pre-processing function with the help of the following string:

> He went to bed at 2 A.M. It\'s way too late! He was only 20% asleep at first, but sleep eventually came.

What can we do with this string?



	Document 1	Document 2	Document 3	⋯	Document *N*
Token 1	0	0	1	62	3
Token 2	0	1	0	61	2
Token 3	1	0	3	101	0
⋯	112	24	38	84	0
Token *M*	2	2	0	12	3
Sum	115	27	42	320	8

Sum
66
64
105
258
19

Figure 27.8: Term-document matrix/ document-term matrix for a hypothetical corpus, with Row Sums and Column Sums.

- Modify every upper case character to its corresponding **lower case** version (avoid if seeking proper nouns and names)

  he went to bed at 2 a.m. it\'s way too late! he was only 20% asleep at first, but sleep eventually came.

- Remove all **punctuation marks** (avoid if seeking emojis):

  iHei went to bed at 2 AM Its way too late He was only 20 asleep at first but sleep eventually cam

- Remove all **numerals** (not ideal when text mining quantities):

  He went to bed at A.M. It\'s way too late! He was only % asleep at first, but sleep eventually came.

- Remove all extraneous **white space**:

  He went to bed at 2 A.M. It\'s way too late! He was only 20% asleep at first, but sleep eventually came.

- Remove characters within **brackets** (and the brackets):

  He went to bed at 2 A.M. It\'s way too late! He was only 20% asleep at first, but sleep eventually came.

- Replace all **numerals with words**:

  He went to bed at two A.M. It\'s way too late! He was only twenty% asleep at first, but sleep eventually came.

- Replace **abbreviations**:

  He went to bed at 2 AM Itś way too late! He was only 20% asleep at first, but sleep eventually came.

- Replace **contractions** (avoid if seeking non-formal speech):

  He went to bed at 2 A.M. It is way too late! He was only 20% asleep at first, but sleep eventually came.

- Replace **symbols with words**:

  He went to bed at 2 A.M. Itś way too late! He was only 20 percent asleep at first, but sleep eventually came.

We typically also remove **stop words** ("a", "an", "the", etc.) and **uninformative words** (which tend to be highly context-dependent), as these unnecessarily increase the number of columns in the DTM.[9] We also usually **stem words** and **complete the stems** to remove unnecessary variation in the text: "sleepful", "sleeping", "sleeps", "slept" all **convey**

9: See the **curse of dimensionality**, Chapter 23.

the meaning of "sleep" and might as well be replaced by the latter term (which is a completed stem or a **lemma**).[10]

10: But there are complications, as we will discuss in Chapter 32: in "operations research", "operating system" and "operative dentistry", the stem "operati" has **different meanings**.

In the BoW approach, then, the text string on which we have been working could be pre-processed to:

> he go bed 2 am way late he 20 percent sleep first sleep eventually come

Note that this is not the only reasonable BoW preparation – as always, **context matters**.

27.2.4 Text Statistics

The problem of how to represent a corpus as BoW DTM is simple to solve, but it requires analysts to make use of their agency.

Consider a **corpus** $\mathscr{C} = \{d_1, \ldots, d_N\}$ consisting of N **documents**, with a **BoW dictionary** $\mathscr{D}_{\mathscr{C}} = t_1, \ldots, t_M$ consisting of M distinct **terms**.[11] For instance, if the corpus is

11: For $i = 1, \ldots, N$, the ith document's dictionary $\mathscr{D}_{\mathscr{C};i} = \{t_{i;1}, \ldots, t_{i;M_i}\}$ consists of the distinct terms of $\mathscr{D}_{\mathscr{C}}$ found in d_i.

$\mathscr{C} = \{$"the dogs who have been let out", "who did that", "my dogs breath smells like dogs food"$\}$,

then $N = 3, d_1 =$ "the dogs who have been let out", $d_2 =$ "who did that", and $d_3 =$ "my dogs breath smells like dogs food", $M_1 = 7$, $M_2 = 3$, $M_3 = 7$, $M = 14$, and the BoW dictionary terms are:

- $t_1 =$ "been",
- $t_2 =$ "breath",
- $t_3 =$ "did",
- $t_4 =$ "dogs",
- $t_5 =$ "food",
- $t_6 =$ "have",
- $t_7 =$ "let",
- $t_8 =$ "like",
- $t_9 =$ "my",
- $t_{10} =$ "out",
- $t_{11} =$ "smells",
- $t_{12} =$ "that",
- $t_{13} =$ "the", and
- $t_{14} =$ "who".

We could further pre-process the corpus (remove stopwords, stem the words, etc.), but for the purposes of illustrating text statistics, we will leave the documents as they are.

The purest bag of word information about a term t in a document d is the raw **term frequency count**

$$tf_{t,d} = \#\text{ times } t \text{ occurs in } d,$$

but its relative usefulness is impacted by the documents' sizes.[12]

12: And size variation among documents.

$\mathbf{tf}_{t,d}$		t													
		1 been	2 breath	3 did	4 dogs	5 food	6 have	7 let	8 like	9 my	10 out	11 smells	12 that	13 the	14 who
d	1	1	0	0	1	0	1	1	0	0	1	0	0	1	1
	2	0	0	1	0	0	0	0	0	0	0	0	1	0	1
	3	0	1	0	2	1	0	0	1	1	0	1	0	0	0

The **relative term frequency** (or term proportion)

$$tf^*_{t,d} = \frac{tf_{t,d}}{M_d}$$

typically provides a more useful representation of the BoW.

$\mathrm{tf}^*_{t,d}$		t													
		1 been	2 breath	3 did	4 dogs	5 food	6 have	7 let	8 like	9 my	10 out	11 smells	12 that	13 the	14 who
d	1	1/7	0	0	1/7	0	1/7	1/7	0	0	1/7	0	0	1/7	1/7
	2	0	0	1/3	0	0	0	0	0	0	0	0	1/3	0	1/3
	3	0	1/7	0	2/7	1/7	0	0	1/7	1/7	0	1/7	0	0	0

At a simpler level, we could also look at the **document frequency** df_t, which is to say, the number of documents in which the term t occurs. To compare a term's usage **across different corpora**, however, it might be preferable to compute the **relative document frequency**

$$df^*_t = \frac{df_t}{N}.$$

This text statistic is only of limited usefulness if N is "too small".

df^*_t	t													
	1 been	2 breath	3 did	4 dogs	5 food	6 have	7 let	8 like	9 my	10 out	11 smells	12 that	13 the	14 who
	1/3	1/3	1/3	2/3	1/3	1/3	1/3	1/3	1/3	1/3	1/3	1/3	1/3	2/3

Another approach is to use the **term frequency-inverse document frequency** (tf-idf) of term t in document d:

$$tf\text{-}idf^*_{t,d} = -tf^*_{t,d} \times \ln df^*_t.$$

This text statistic is a **heuristic**; although it has no solid theoretical backing, it is nevertheless commonly-used.[13]

13: Silge (an early backer of tf-idf) and Schnoebelen suggest an alternative in the form of weighted log odds, which can also be used with non-text data.

The rationalization for its use is that if most of the documents contain the term t, then $df^*_t \approx 1$ and the presence of that term in a document does not provide a lot of information about said document (since it shows up in most documents):

$$tf\text{-}idf^*_{t,d} \approx -tf^*_{t,d} \times \ln 1 = 0.$$

Furthermore, if the term t does not occur often in a document d for which M_d is large, then $tf^*_{t,d} \approx 0$ and

$$tf\text{-}idf^*_{t,d} \approx -0 \times \ln df^*_t = 0.$$

In this BoW approach, it is the terms that appear relatively often **only in a small subset** of documents (with large values) that are crucial to understanding those documents in the general context of the corpus.

$\text{tf}-\text{idf}_t^*$		t													
		1 been	2 breath	3 did	4 dogs	5 food	6 have	7 let	8 like	9 my	10 out	11 smells	12 that	13 the	14 who
d	1	0.16	0	0	0.06	0	0.16	0.16	0	0	0.16	0	0	0.16	0.06
	2	0	0	0.37	0	0	0	0	0	0	0	0	0.37	0	0.14
	3	0	0.16	0	0.12	0.16	0	0	0.16	0.16	0	0.16	0	0	0

27.2.5 Text Visualization

One of the major differences between **text analysis** and plain numerical **data analysis** is that even though we are able to interpret numerical results with some (often minimal) effort, we can easily interpret text analysis results with **no effort**: we have a built-in **semantics detector** – text is not just a label for data, it has **meaning** (derived from the context) that is automatically available to us.[14]

In truth, we can train ourselves to read numerical data and results, especially with the help of **data visualizations** (see Chapter 18 and [9], for instance). Somewhat paradoxically, we can also visualize text data. Common methods include **barcharts**, **scatterplots**, **word clouds**, and **phrase nets** (see Figure 27.9 and Section 27.4).[15]

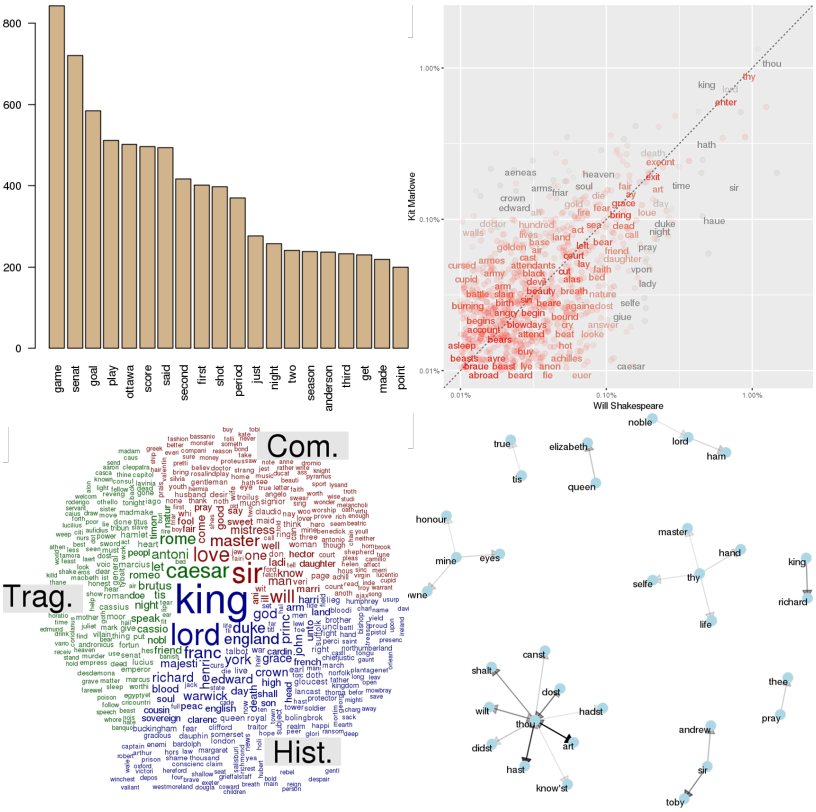


Figure 27.9: Examples of text visualizations: barchart (top left, from Section 27.4.1), scatterplot (top right, from Section 27.4.2), word cloud (from Section 27.4.3), and phrase net (from Section 27.4.2).

14: My son Llewellyn, upon learning how to read, complained that he **couldn't help but read** text when he saw it – the blissful ignorance of the past is forever gone.

15: These can be quite handy when conducting a BoW analysis on a subject of which the analysts know very little, or if the text is in a language that they do not master yet.

27.3 Text Mining Tasks

> "If the computer can successfully tell a joke as well as Henry Youngman, then that's the voice I want." [Roger Ebert, TED Talk, 2011]

We can easily leverage **machine learning** (ML) **techniques** to improve the BoW (this section) and semantics (see Chapter 32) approaches to text analysis.

We have seen that text usually enters the text analysis pipeline in **unstructured** and **unorganized** formats, from a variety of sources. Through pre-processing, text becomes **clean** (yet remains unstructured).

The BoW approach provides a framework for a **structured numerical representation** of text data, either in the form of DTM/TDM or **tidy data** (see Section 27.4 for examples of the latter). It is on these representations that ML algorithms are unleashed.

At the ML stage, it is important to remember where the data comes from and the context in which it applies. The **text mining/NLP pipeline** of Figure 27.10 applies in most (if not all) text analysis situations.

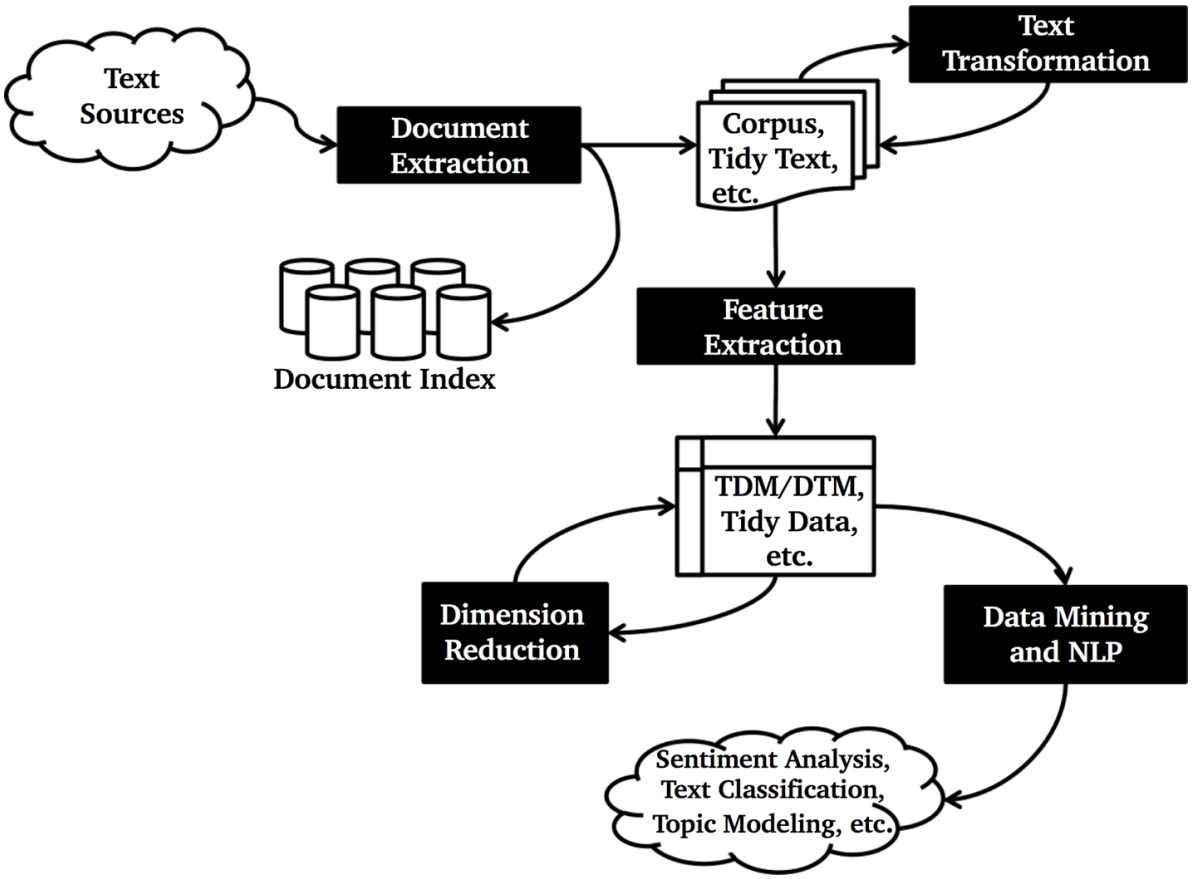


Figure 27.10: Text mining and NLP pipeline [author unknown].

We have discussed **document extraction** in Chapter 16, **feature extraction** in Chapter 23, and **machine learning** in Chapters 19–22.

In **supervised learning** (SL), there is a target/response against which to **train** and **test** the models; typical tasks include **classification** and **class probability estimation**, **regression** and **value estimation**, etc.

In **unsupervised learning** (UL), there is no target; UL could be used to **discover** potential target/response levels that could eventually be used in SL tasks with new data, say; typical tasks include **association rules** and **hypothesis discovery**, **similarity matching** and **clustering**, etc.

Other ML tasks include **profiling**, **link prediction**, and so on. In text mining, the most commonly used ML tasks are **classification** and **clustering**.

27.3.1 Classification

In **classification**, a sample set of data (the **training set**) is used to determine **rules** (or a **model**) that can be used to **divide** the data into **pre-determined groups** (also known as **classes**). The model is then **validated** by examining its performance on a **test set**.

In **text classification**, the data must first be given a **numerical representation** (DTM/TDM/tidy data) – it is on this object that **classifiers** are trained.

For instance, we may wish to answer questions such as:

- Based on the terms that appear in a text, can we determine who its author is likely to be?
- Based on the words that appear in a news story, is it propaganda?
- Is the email that was just received legitimate or malicious?
- Should a city trigger its emergency response system based on social media conversations?
- What is a tweet's main sentiment?
- etc.

Text Classification Workflow In particular, the text classification process should follow the regular classification pipeline, with the exception of the conversion from text data to numerical representation:

1. data collection;
2. data pre-processing;
3. exploration and text visualization;
4. text representation;
5. training the model, and
6. testing and evaluation.

Notes and Comments Remember that in order to train and test a classifier, the true labels have to be known for at least some of the data – this might not be achieved easily.[16]

Classification is also affected by the **No Free-Lunch Theorem** stating, in effect, that no single classifier is always the best option – we have to consider a number of models, on a case-by-case basis.

16: It is usually quite costly to obtain these labels, especially with large text or image datasets.

In situations where at least one of the class labels **occurs rarely** in the training data, the classifiers may be swamped by the frequent labels: how would an e-mail spam filter handle a term it has never encountered before? This hurdle is tricky to overcome, technically-speaking, especially as rare occurrences are often more interesting and/or important in the problem context.[17]

17: We discuss these briefly in Section 21.3.

Since it is recommended that we try out different classifiers, how can we determine if a model is preferable to another? The theory of **performance metrics** is richer for **binary classifiers** than general classifiers: ideally, a good model would have high rates of true positives and true negatives, and low rates both of false positives and false negatives.[18]

18: There are complications, as expected: see Sections 19.4.4 and 21.1.2 for more details.

Multinomial naïve Bayes Classification We will showcase a classification approach which has found quite a bit of success as an e-mail spam filter.[19] **Multinomial naïve Bayes** is a classifier for which the **feature vectors** in each class are assumed to have a **multinomial distribution**.

19: It is a variant of the algorithm presented in Section 21.4.4.

Consider a training set n of email messages – the **records**. Each record has ℓ features, the **frequencies** (or relative frequencies) of ℓ pre-selected terms in the email message body. Each record can then be represented by its signature $\mathbf{x} = (x_1, \ldots, x_\ell)$.

We assume that there are K categories in which a record could be classified.[20] Let $\{C_k \mid k = 1, \ldots, K\}$ denote the categories.

20: Perhaps the class labels are: spam, quarantined, personal, business, etc.

For any incoming e-mail message, the **classification problem** is to determine the **posterior distribution**

$$P(\mathbf{x} \in C_k \mid x_1, \ldots, x_\ell)$$

for each label k. The **predicted class** of $\mathbf{x}$ is the class C_k for which the posterior is largest.

Fix k. From **Bayes' Theorem**, we have

$$P(\mathbf{x} \in C_k \mid x_1, \ldots, x_\ell) \propto P(\mathbf{x} \in C_k) \times P(x_1, \ldots, x_\ell \mid \mathbf{x} \in C_k).$$

The **naïve assumption** is that

$$P(x_1, \ldots, x_\ell \mid \mathbf{x} \in C_k) = P(x_1 \mid \mathbf{x} \in C_k) \times \cdots P(x_\ell \mid \mathbf{x} \in C_k),$$

so that

$$P(\mathbf{x} \in C_k \mid x_1, \ldots, x_\ell) \propto P(\mathbf{x} \in C_k) \times \prod_{i=1}^{\ell} P(x_i \mid \mathbf{x} \in C_k).$$

The **multinomial assumption** is that

$$P(x_i \mid \mathbf{x} \in C_k) = p_{k,i}^{x_i},$$

where $p_{k,i} \in [0, 1]$ for each feature (word) $1 \leq i \leq \ell$.

Combining these assumptions, the posterior "probabilities" are then

$$P(\mathbf{x} \in C_k \mid x_1, \ldots, x_\ell) \propto P(\mathbf{x} \in C_k) \times \prod_{i=1}^{\ell} p_{k,i}^{x_i}.$$

The model can be further **linearized** by taking logarithms on both sides:

$$\log P(\mathbf{x} \in C_k \mid x_1, \ldots, x_\ell) \propto b_k + \sum_{i=1}^{\ell} x_i \log p_{k,i}.$$

The classifier is trained by **estimating the parameters** $p_{k,i}$ on a subset of all records and by specifying the "priors" b_k. The predicted class $C(\mathbf{x})$ is the C_k for which $b_k + \sum_{i=1}^{\ell} x_i \log p_{k,i}$ is **maximized**.[21]

21: Recall that this classifier is **not calibrated** – the relative values of the posterior "probabilities" have no intrinsic value in and of themselves.

If a message encounters terms (tokens, words) that were not seen in the training data, it is impossible to predict its most likely class membership using (non-existent) past behaviour. In that case, to avoid divisions by 0, we make use of the **corrected estimate**

$$\hat{p}_{k,i} = \frac{\sum_{\mathbf{x} \in C_k} x_i + 1}{\sum_{\mathbf{x} \in C_k} (x_1 + \cdots + x_\ell) + |v|} = \frac{(\#w_i \in C_k) + 1}{W_k + |v|},$$

where $|v|$ is the size of the vocabulary, $\#w_i \in C_k$ is the **frequency** of the word w_k in the training documents belonging to class C_k, and W_k is the **count of all words** appearing in training documents in class C_k.

As an example, consider the training set and testing set below (raw and processed), describing the sentiment (class: + or −) associated with 6 reviews for a specific phone.

Training set

cl	ID	text
+	i1	I love this phone
+	i2	amazing sound quality
+	i3	Love this great phone
-	i4	i hate it
-	i5	bad quality
-	i6	so bad Hate it

Testing set

??	i7	hate hate HATE the phone quality

Processed training set

cl	ID	text
+	i1	love phone
+	i2	amazing sound quality
+	i3	love great phone
-	i4	hate
-	i5	bad quality
-	i6	bad hate

Processed testing set

??	i7	hate hate hate phone quality

For the priors of each class, we use the proportions of positive and negative reviews in the training data. In the processed data, there are 8 distinct vocabulary terms, and there are 8 (resp. 5) terms in the positive (resp. negative) reviews. The corrected estimates for the vocabulary word "amazing" are computed below.

$$P(+) = \frac{3}{6} = 0.5; \quad P(-) = \frac{3}{6} = 0.5 \implies b_+ = b_- = \ln 0.5;$$

$$|v| = 8, W_+ = 8, W_- = 5;$$

$$\hat{p}_{+,\text{amazing}} = \frac{(\#\text{amazing} \in +) + 1}{W_+ + 8} = \frac{1+1}{8+8} = \frac{1}{8}$$

$$\hat{p}_{-,\text{amazing}} = \frac{(\#\text{amazing} \in -) + 1}{W_- + 8} = \frac{0+1}{5+8} = \frac{1}{13}$$

The corrected estimates for each vocabulary word in the training documents are shown below, as is the signature vector **x** for the lone test record.

Training set

$\widehat{p}$	amazing	bad	great	hate	love	phone	quality	sound
+	0.1250	0.6025	0.1250	0.0625	0.1875	0.1875	0.1250	0.1250
−	0.0769	0.2308	0.0769	0.2308	0.0769	0.0769	0.1538	0.0769

Testing set

	amazing	bad	great	hate	love	phone	quality	sound
i7	0	0	0	3	0	1	1	0

Simple computations show that

$$P(+ \mid \mathbf{x})\text{“}=\text{”}\ 2.9 \times 10^{-6} < P(- \mid \mathbf{x}) = 9.7 \times 10^{-6},$$

from which we conclude that the test review is **negative**.[22]

22: Hopefully, that is not much of a surprise in light of the original test review.

Of course, any classification algorithm may be used. Common methods also include support vector machines and artificial neural networks (see Chapter 21), not only multinomial naïve Bayes. We will have more to say on the topic in Section 27.4.4.

27.3.2 Clustering

The aim of **clustering** is to **divide** the data into **latent groups** (also known as **clusters**). Within a cluster, data points are seen as **similar** to one another; between clusters, they are seen as **dissimilar**. As befits the unsupervised learning nature of the task, the cluster labels are **not pre-determined**.

For instance, we may wish to:

- divide existing social media users into subgroups based on the shared characteristics of their posts;
- create (new) taxonomies on the fly, as new items are added to a group of items to ease product navigation;
- cluster terms within a corpus of document (topic modeling);
- cluster documents within a corpus based on their use of terms;
- identify keywords in a document;
- etc.

Text Clustering Workflow The steps are quite similar to those of text classification:

1. data collection;
2. data pre-processing;
3. exploration and text visualization;
4. text representation;
5. run multiple clustering algorithms with parameter variations, and
6. compare and validate the results.

Notes and Comments Conceptually, clustering is relatively intuitive for people: we recognize clusters when we see them.

But there are issues, chiefly:

- there is no agreed-upon definition of what a cluster is;
- there is no "magic" recipe to determine which similarity measure to use;
- the number of cluster is not usually specified, and
- due to the non-deterministic nature of many clustering algorithms, the results are often unreplicable.

And what does it mean to cluster **text data**? With a DTM text representation, we can **cluster the documents with respect to the terms**, which is to say that we look for documents that have **similar term signatures**. With a TDM text representation, we can **cluster the terms with respect to the documents**, which is to say that we look for terms that have **similar document signatures**.

We will have more to say on the topic in Section 27.4.5.

27.3.3 Sentiment Analysis

Most of us have a good native understanding of the emotional intent of words, which leads us to infer **surprise**, **disgust**, **joy**, **pain**, and so on from a text segment.[23] When applied by machines to a block of text, the (somewhat subjective) process of identifying emotions is known as **sentiment analysis** (or **opinion mining**).

23: Although **sarcasm** or **lies** are not always obvious without other contextual cues.

Typical sentiment questions could include:

- Is this movie review positive or negative?
- Is this customer email a complaint?
- How have newspapers' attitudes about the PM changed since the election?
- etc.

Most humans would typically be able to answer these questions when presented with the appropriate text documentation, but there is no guarantee that each individual's reading of the situation would be the same. For text processing machines (even modern LLMs), questions of this nature may be quite difficult to answer.

Challenges Data analysis is not easy, in general, but sentiment analysis is even more complicated, as:

- the **topic** may change halfway through the text;
- the author may be using **rhetorical devices** ;
- we do not always agree on the **emotional content** of text (due to cultural context, or lack of familiarity with the language, or different political affiliations, etc.);
- words may have **different meanings**/**emotional values** depending on the context (stolen goods may be **hot**, in which case we would want nothing to do with them; a new song may be **hot**, in which case we would probably stream it right away);

- **qualifiers** can drastically change a term's emotional value (note the emotional difference between **he was really happy today** vs. **he was not really happy today**, which may only differ in one term, but has a completely different meaning), etc.

Sentiment analysis is a **supervised learning** problem, requiring dictionaries of **emotional content** to have been compiled ahead of time or **sentiment labels** to have been assigned to a training text dataset.[24] The following reviews were taken from Amazon: there is a 5−star, a 3−star, and a 1−star review – but which is which?

> **Review 1:** "Love the jeans, price, fit, but even more, love the suppliers. Simple concerns were not only answered immediately, they went beyond any expectations I had! Will definitely be buying through this supplier, highly recommended!"
>
> **Review 2:** "DON'T BUY. Great series aside, this special addition is pathetic. They're basically mass-market paperbacks: small and uncomfortable to hold. The regular paperback versions are far superior for basically the same price."
>
> **Review 3:** "Beginning the second use, the bowl keeps falling out 30 seconds after the mixing starts. A bit disappointed."

This is how the AIPlaybook at a16z.com/ai saw the situation in 2021:

"Love the jeans, price, fit, but even more, love the suppliers. Simple concerns were not only answered immediately, they went beyond any expectations I had! Will definitely be buying through this supplier, highly recommended!"

Scores

API Name	Result	Total request time	API Time
+ Sentiment.JS (node.js library)	very positive (90)	299	0
+ Sentimental (node.js library)	very positive (90)	289	0
+ IBM Alchemy Language API	-1	341	43
+ IBM Watson Developer Cloud	positive (72)	1472	1128
+ Google Cloud APIs	-1	651	351
+ Microsoft Azure Cognitive Services	very positive (93)	789	482

"DON'T BUY. Great series aside, this special addition is pathetic. They're basically mass-market paperbacks: small and uncomfortable to hold. The regular paperback versions are far superior for basically the same price."

Scores

API Name	Result	Total request time	API Time
+ Sentiment.JS (node.js library)	neutral (10)	163	0
+ Sentimental (node.js library)	neutral (10)	157	1
+ IBM Alchemy Language API	-1	240	79
+ IBM Watson Developer Cloud	neutral (-3)	1164	922
+ Google Cloud APIs	-1	436	218
+ Microsoft Azure Cognitive Services	negative (-69)	788	609

24: Related tasks include:

- discarding subjective information (**information extraction**);
- recognizing opinion-oriented questions (**question answering**);
- accounting for multiple viewpoints (**summarization**);
- identifying suitability of videos for kids;
- identifying bias in news sources;
- identifying appropriate content for ad placement, etc.

"Beginning the second use, the bowl keeps falling out 30 seconds after the mixing starts. A bit disappointed."

Scores

API Name	Result	Total request time	API Time
+ Sentiment.JS (node.js library)	negative (-30)	65	0
+ Sentimental (node.js library)	negative (-30)	66	0
+ IBM Alchemy Language API	-1	379	211
+ IBM Watson Developer Cloud	neutral (-6)	1300	1026
+ Google Cloud APIs	-1	408	281
+ Microsoft Azure Cognitive Services	very negative (-78)	684	486

There is a fair amount of variation, both within and across reviews. What would a modern tool like GPT make of these?

Patrick: Good evening, GPT. I am going to post three product reviews. One is a 5-star review, one is a 3-star review, and one is a 1-star review. I would like you to tell me which review matches with which score.

GPT: Absolutely, that sounds like an interesting exercise! Go ahead and post the three reviews, and I'll do my best to match each review with the correct star rating.

Review 1: "Love the jeans, [...]"

Review 2: "DON'T BUY. [...]"

Review 3: "Beginning [...]"

Based on the content and tone of each review, here's how I would match them with their respective star ratings.

Review 1: This review is overwhelmingly positive, emphasizing satisfaction with both the product (jeans) and the supplier's customer service. The reviewer mentions loving the product and having their expectations exceeded, and they even recommend the supplier. This review aligns with a **5-star rating**.

Review 3: This review expresses disappointment, but it's not entirely negative. The reviewer seems to have had a specific issue with the product (a bowl falling out during use). However, they don't express complete dissatisfaction or advise against purchasing the product. This seems more in line with a **3-star rating** – not completely dissatisfied, but not fully satisfied either.

Review 2: This review is clearly negative. The reviewer strongly advises against purchasing the product, using phrases like "DON'T BUY" and describing the product as "pathetic." They also compare it unfavorably to another version, suggesting the other is a better choice. This tone and content are indicative of a **1-star rating**.

Let me know if you agree with these assessments!

Thank you, I think that this is indeed what intelligent agents would conclude.

For the sake of completeness, I should mention that Review 1 was the 5-star review; Review 2, the 3-star review, and Review 3, the 1-star review. There is nothing you could have done differently to obtain these results... humans are not always logical when it comes to rating things.

Thank you for sharing the actual ratings, and you're absolutely right! Human reviews can be quite subjective and not always follow a logical pattern. People's rating criteria can vary greatly, and what might be a minor inconvenience to one person could be a major drawback to another. It's an interesting reminder of the diversity in human experiences and expectations.

Perhaps GPT's ratings were skewed because it was told that one of the reviews was a 5-star review, and so on. What if we prompt it differently, in a new conversation? As it turns out, GPT comes to the same conclusion, as we can see below.

Patrick: Good evening, GPT. I am going to provide you with 3 product reviews. I would like for you to rate them, on a scale from 1 (poor) to 5 (great).

GPT: Sure, I'd be happy to help with that. Please go ahead and share the three product reviews, and I'll rate each of them on a scale from 1 to 5 based on their content.

Review 1: "Love the jeans, [...]"

Review 2: "DON'T BUY. [...]"

Review 3: "Beginning [...]"

Based on the content of each review, here are the ratings I would assign.

Review 1: This review is very positive. The customer expresses satisfaction with multiple aspects of the product (jeans) and exceptional satisfaction with the supplier's customer service. The enthusiasm and recommendation suggest a high level of satisfaction. I would rate this as a **5 (great)**.

Review 2: This review is highly negative. The customer advises against purchasing the product and describes it as "pathetic," indicating significant dissatisfaction. The comparison with a superior version of the product emphasizes their disappointment. I would rate this as a **1 (poor)**.

Review 3: This review expresses a specific problem and disappointment, but it doesn't completely condemn the product. It seems like a moderate level of dissatisfaction, possibly due to a single but significant issue. I would rate this as a **3 (average)**, assuming the problem is not widespread or indicative of overall poor quality.

These ratings are based on the tone and content of the reviews, reflecting the customers' levels of satisfaction or dissatisfaction with their purchases.

Thank you!

Term-by-Term and Document-by-Document Sentiment Analysis In this chapter, we differentiate 2 types of sentiment analyses:

- **term-by-term** (TBT) sentiment analysis looks at the emotional content of **individual tokens** (terms in the text) and deduces an **emotional score** for passages containing them;
- **document-by-document** (dbd) sentiment analysis looks at **scored passages** and tries to find tokens which **carry the emotional load** to predict how a new passage would score on some emotional spectrum.[25]

25: Note that these terms are not widely-used in the analyst community.

TBT is not a complicated technical task: it only requires the ability to match a **lexicon score** to each term, and to add the scores; DBD is basically a **classification problem**.

Sentiment Lexicons TBT sentiment analysis relies heavily on **lexicons**, which is to say, lists of terms which have been **ranked** (by experts) on some **emotional scale**. Commonly-used lexicons include:

- **AFINN:** words are ranked on a scale from -5 (negative) to 5 (positive);
- **BING:** words are given a binary score (negative/positive);
- **NRC:** words are assigned category(ies) of sentiments;
- **LOUGHRAN:** words are placed in categorical bins.

"abandon"

- AFINN: -2
- BING: NA
- NRC: fear, negative, sadness
- LOUGHRAN: negative

"bad"

- AFINN: -3
- BING: negative
- NRC: anger, disgust, fear, etc.
- LOUGHRAN: negative

"not"

- AFINN: NA
- BING: NA
- NRC: NA
- LOUGHRAN: NA

"egregious"

- AFINN: ?
- BING: ?
- NRC: ?
- LOUGHRAN: ?

What's the best lexicon to use? As always, **context matters**. Is there any reason to expect the various lexicons to give the same scores? Each of these lexicons contains a majority of negative terms (keeping in mind that most words in the English language are **neutral**), so there could at least be some correlation.

Once a lexicon has been selected, TBT is simply a matter of **chunking** the text and computing sentiment scores on each block (every 100 words, every 100 lines, every chapter, etc.). Does the sectioning approach matter? Again, context matters.

We provide examples of TBT and DBD in Sections 27.4.6 and 27.4.8. For a more in-depth discussion on text mining and natural language processing, interested readers are advised to also consult Chapter 32 and [40, 39, 24, 3, 1, 25, 8].

27.4 Examples

Various concepts of text analysis and text mining are illustrated using `R` and `Python` in the following examples, some of which are inspired by the excellent [39] and [26].

27.4.1 NHL Game Recaps

In this example, we introduce the basic notions of text mining using the `tm` (text mining) and `qdap` (quantitative discourse analysis package) libraries in `R`.

The main dataset we work with is the text content of Associated Press game recaps involving the Ottawa Senators during the 2016-2017 NHL season.[26]

26: All of this section's datasets are available at github.com/potrbollvy/Data-Training.

Initializing the Environment

```
install.packages("tm")
install.packages("qdap")
```

Preliminaries

We start with a simple example to showcase the possibilities.

```
new_text <- "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."
```

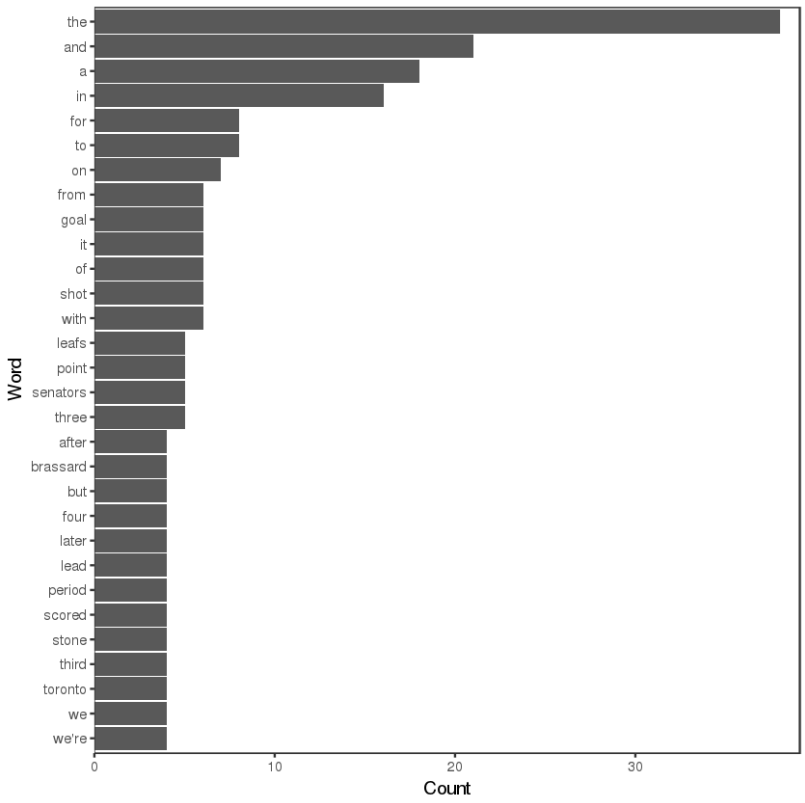
We find the 20 most frequent terms using `qdap`'s `term_count()`.

```
(term_count <- qdap::freq_terms(new_text,20))
```

```
   WORD     FREQ       WORD     FREQ       WORD     FREQ       WORD     FREQ
1  the        38    9  goal        6    17 three       5    25 scored      4
2  and        21    10 it          6    18 after       4    26 stone       4
3  a          18    11 of          6    19 brassard    4    27 third       4
4  in         16    12 shot        6    20 but         4    28 toronto     4
5  for         8    13 with        6    21 four        4    29 we          4
6  to          8    14 leafs       5    22 later       4    30 we're       4
7  on          7    15 point       5    23 lead        4
8  from        6    16 senators    5    24 period      4
```

There are more than 20 entries because of ties (at 4 occurrences apiece). This information can also be displayed as a chart.

```
plot(term_count)
```



Sens Recaps Data

We now import the data for all games during the season.

```
recaps <- read.csv(file="Recap_data.csv", header=TRUE, sep=",", stringsAsFactors=FALSE)
nrow(recaps)
str(recaps)
```

```
[1] 101

'data.frame':  101 obs. of  34 variables:
 $ GP          : int  1 2 3 4 5 6 7 8 9 10 ...
 $ X0_Type     : chr  "1_Regular" "1_Regular" "1_Regular" "1_Regular" ...
 $ Date        : chr  "10/12/2016" "10/15/2016" "10/17/2016" "10/18/2016" ...
 $ Time        : chr  "7:00 PM" "7:00 PM" "7:30 PM" "7:30 PM" ...
 $ X           : chr  "" "" "A" "" ...
 $ Opponent    : chr  "Toronto Maple Leafs" "Montreal Canadiens" "Detroit Red Wings" ...
 $ GF          : int  5 4 1 7 1 3 2 2 2 1 ...
 $ GA          : int  4 3 5 4 4 0 5 0 1 0 ...
 $ Result      : chr  "W" "W" "L" "W" ...
 $ OT_SO       : chr  "OT" "SO" "" "" ...
```

```
 $ W_Record    : int  1 2 2 3 3 4 4 5 6 7 ...
 $ L_Record    : int  0 0 1 1 2 2 3 3 3 3 ...
 $ OL_Record   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ Streak      : chr  "W 1" "W 2" "L 1" "W 1" ...
 $ OTT_S       : int  30 38 32 42 28 28 33 22 32 24 ...
 $ OTT_PIM     : int  13 10 22 14 8 2 4 11 13 20 ...
 $ OTT_PPG     : int  0 0 0 1 0 0 2 0 0 0 ...
 $ OTT_PPO     : int  2 4 3 2 3 1 4 2 2 4 ...
 $ OTT_SHO     : int  0 0 1 1 0 0 0 0 0 0 ...
 $ OPP_S       : int  38 24 25 35 35 22 19 37 33 27 ...
 $ OPP_PIM     : int  11 10 20 6 6 2 8 9 11 22 ...
 $ OPP_PPG     : int  0 1 2 1 2 0 0 0 0 0 ...
 $ OPP_PPO     : int  4 4 4 5 4 1 2 4 3 2 ...
 $ OPP_SHG     : int  0 0 0 0 0 0 0 0 0 0 ...
 $ ATT         : chr  "17,618" "18,195" "20,027" "11,061" ...
 $ LOG         : chr  "2:36" "2:44" "2:33" "2:43" ...
 $ AP_Headline : chr  "Maple Leafs\xcd Matthews has modern record" ...
 $ AP_Recap    : chr  "Auston Matthews needed 40 minutes to get into"| __truncated__ ...
 $ SSS_Author  : chr  "Ross A" "Ary M" "Michaela Schreiter" "Ary M" ...
 $ SSS_Headline: chr  "Auston Matthews Loses 5-4 to Sens in OT" ...
 $ SSS_Recap   : chr  "The NHL.com headline for the game was \xd2Auston Matthews scores"| __truncated__
 $ OPP_Blog    : chr  "Pension Plan Puppet" "Eyes on the Prize" ...
 $ OPP_Title   : chr  "Sens 5, Auston Matthews 4 (OT)" ...
 $ OPP_Recap   : chr  "The first period started exactly the way that Leafs' fans wanted"| __truncated__
```

It is child's play to isolate the text from individual game recaps.

```
AP.recaps <- recaps$AP_Recap
head(AP.recaps,2)
```

```
'Auston Matthews needed 40 minutes to get into the NHL record book. In the highest-scoring debut
  in modern NHL history, Matthews scored four goals for the Toronto Maple Leafs, but Kyle Turris
  scored 37 seconds into overtime to give the Ottawa Senators to a 5-4 victory Wednesday night.
  Matthews got his fourth with 3 seconds left in the second  period, bringing his mother to
  tears in the stands. He called it a \xf1surreal\xee moment, adding that \xf1I couldn\xcdt
  believe that was happening out there.\xee [...] UP NEXT Maple Leafs: Host Boston on Saturday
  night for their home opener. Senators: Host Montreal on Saturday night.'
'Guy Boucher trusted his instincts when selecting skaters for the shootout and it paid off for the
  Ottawa Senators. The Senators\' head coach opted to go with defenseman Erik Karlsson and the
  captain scored the game winner to give Ottawa a 4-3 victory over the Montreal Canadiens on Saturday
  night to open the season with back-to-back wins. \'\'Sometimes it\'s just small things  and you
  follow a gut feeling,\'\' Boucher said. [...]'
```

There are odd characters in the game recaps (`xf1`, `xee`, etc.), which highlight some issue with text encoding and formatting. We revisit the last few steps with a slightly different data file.

```
recaps <- read.csv(file="Recap_data_first_pass.csv", header=TRUE,
                   sep=",", stringsAsFactors=FALSE)
AP.recaps <- recaps$AP.recaps
head(AP.recaps,2)
```

```
'Auston Matthews needed 40 minutes to get into the NHL record book. In the highest-scoring debut
  in modern NHL history, Matthews scored four goals for the Toronto Maple Leafs, but Kyle Turris
  scored 37 seconds into overtime to give the Ottawa Senators to a 5-4 victory Wednesday night.
  Matthews got his fourth with 3 seconds left in the second period, bringing his mother to tears
  in the stands. He called it a "surreal" moment, adding that "I couldn\'t believe that was
  happening out there." [...]'
'Guy Boucher trusted his instincts when selecting skaters for the shootout and it paid off for the
  Ottawa Senators. The Senators\' head coach opted to go with defenseman Erik Karlsson and the captain
  scored the game winner to give Ottawa a 4-3 victory over the Montreal Canadiens on Saturday night
  to open the season with back-to-back wins. "Sometimes it\'s just small things and you follow a gut
  feeling," Boucher said. [...]'
```

The results are easier to read, for sure, but for reasons that are too technical to get into here, the encoding of `Recap_data_first_pass.csv` creates issues with `tm` and `qdap` down the road, but the issues disappear when we use a different encoding (UTF-8).

```
recaps <- read.csv(file="Recap_data_first_pass_utf8.csv", header=TRUE, sep=",",
                   stringsAsFactors=FALSE)
AP.recaps <- recaps$AP.recaps
```

VCorpus from Vector with `tm`

The `tm` package makes it easy to work with vector sources and volatile corpora. For instance, we can make a **vector source** as follows.

```
AP.recaps.source <- tm::VectorSource(AP.recaps)
```

This vector source can be converted to a **volatile corpus**.

```
AP.recaps.corpus <- tm::VCorpus(AP.recaps.source)
```

At a fundamental level, the volatile corpus contains the following information.

```
AP.recaps.corpus
```

```
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 101
```

This is not entirely useful, to be honest, although we do recover the 101 games played by the Senators during the season. Let's say we wanted more details on the 15th game.

```
AP.recaps.corpus[[15]]
```

```
<<PlainTextDocument>>
Metadata:  7
Content:  chars: 2871
```

There are two entries in the list; the first is the game recap text.

```
AP.recaps.corpus[[15]][1]
```

```
$content = 'For a team playing its third game in four nights, the Minnesota Wild looked plenty
  fresh on Sunday night -- even in overtime. Matt Dumba scored late in the extra session and
  Darcy Kuemper stopped 35 shots, helping Minnesota beat the Ottawa Senators 2-1. The Wild [...]'
```

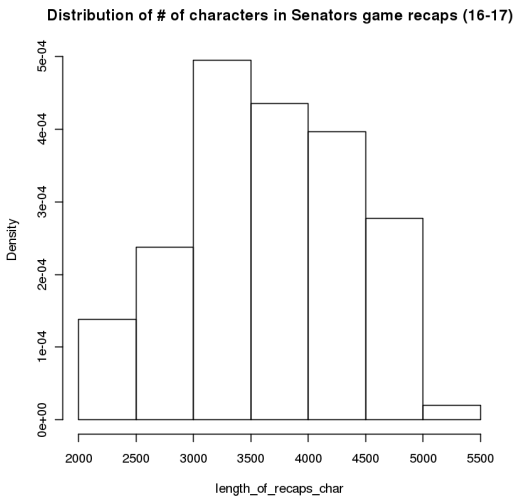
The entry's **metadata** can be queried as follows.

```
AP.recaps.corpus[[15]][2]
```

```
$meta
  author       : character(0)
  datetimestamp: 2019-09-15 13:29:16
  description  : character(0)
  heading      : character(0)
  id           : 15
  language     : en
  origin       : character(0)
```

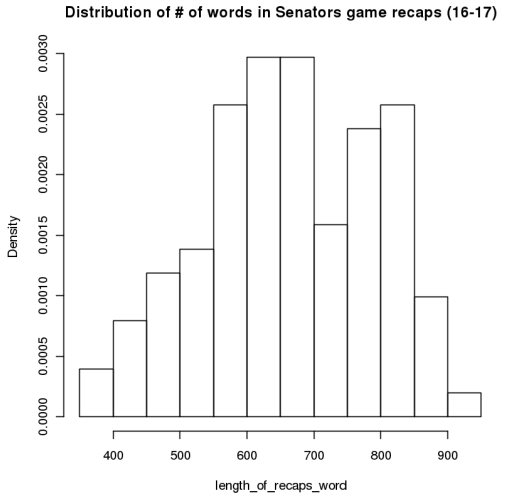
We can also take a look at some basic statistics regarding the **number of characters** and the **number of words** in the game recaps.

```
length_of_recaps_char <- vector(mode="numeric", length=nrow(recaps))
for(j in 1:nrow(recaps)){
    length_of_recaps_char[j]=nchar(AP.recaps.corpus[[j]][1])
}

hist(length_of_recaps_char, freq=F,
     main="Distribution of # of characters in Senators game recaps (16-17)")
summary(length_of_recaps_char)
```




```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2027    3223    3689    3683    4227    5087
```

```
length_of_recaps_word <- vector(mode="numeric", length=nrow(recaps))
for(j in 1:nrow(recaps)){
    length_of_recaps_word[j]=length(strsplit(gsub(' {2,}',' ',
                                        AP.recaps.corpus[[j]][1]),' ')[[1]])
}

hist(length_of_recaps_word, freq=F,
     main="Distribution of # of words in Senators game recaps (16-17)")
summary(length_of_recaps_word)
```




```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    375     565     665     664     774     921
```

Pre-Processing a Document with `tm`

To get the most of this corpus, we must first transform it into a **bag-of-words** (BoW). We first show how to implement the various text processing functionalities on the text string used in Section 27.2.3.

```
(text <- "<i>He</i> went to bed at       2 A.M. It\'s way too late!  He was only 20%
    asleep at first, but sleep eventually came.")
```

```
[1] "<i>He</i> went to bed at       2 A.M. It\'s way too late!  He was only 20% asleep at first,
but sleep eventually came."
```

All characters can be converted to lower case with the `tolower()` function.

```
tolower(text)
```

```
[1] "<i>he</i> went to bed at       2 a.m. it\'s way too late!  he was only 20% asleep at first,
but sleep eventually came."
```

The output of the following three `tm` functions should be clear from their name.

```
tm::removePunctuation(text)
```

```
[1] "iHei went to bed at       2 AM Its way too late  He was only 20 asleep at first
but sleep eventually came"
```

```
tm::removeNumbers(text)
```

```
[1] "<i>He</i> went to bed at        A.M. It\'s way too late!  He was only % asleep at first,
but sleep eventually came."
```

```
tm::stripWhitespace(text)
```

```
[1] "<i>He</i> went to bed at 2 A.M. It\'s way too late! He was only 20% asleep at first,
but sleep eventually came."
```

Pre-Processing a Document with `qdap`

Some of the more sophisticated processes are implemente in `qdap`. The functionality should be clear from the function's name (as well as its output).[27]

27: Note that all of these also strip unnecessary spaces in the string.

```
# Remove text within brackets
qdap::bracketX(text)

# Replace numbers with words
qdap::replace_number(text)

# Replace abbreviations
qdap::replace_abbreviation(text)

# Replace contractions
qdap::replace_contraction(text)

# Replace symbols with words
qdap::replace_symbol(text)
```

```
[1] "He went to bed at 2 A.M. It\'s way too late! He was only 20% asleep at first,
but sleep eventually came."

[1] "<i>He</i> went to bed at two A.M. It\'s way too late! He was only twenty% asleep at first,
but sleep eventually came."

[1] "<i>He</i> went to bed at 2 AM It\'s way too late! He was only 20% asleep at first,
but sleep eventually came."

[1] "<i>He</i> went to bed at 2 A.M. it is way too late! He was only 20% asleep at first,
but sleep eventually came."

[1] "<i>He</i> went to bed at 2 A.M. It\'s way too late! He was only 20 percent asleep at first,
but sleep eventually came."
```

Stopwords

Stopwords are those words that do not carry semantic content, partly because they occur too frequently to really be "seen" by speakers/readers, such as the "said" tag in a novel. They are often removed from the text prior to BoW analysis.

```
# List standard English stop words
tm::stopwords("en")

# Print text without standard stop words
tm::removeWords(text,tm::stopwords("en"))
```

```
[1] "<>He</> went  bed       2 A.M. It's way  late!  He   20% asleep  first,
sleep eventually came."
```

Of course, stopwords may vary from one context to the next,[28] and it is possible to add or subtract words fromt he stopwords list.

28: Or from one language to the next.

```
# Add "sleep" and "asleep" to the list: new_stops
new_stops <- c("sleep","asleep",tm::stopwords("en"))

# Remove stop words from text
tm::removeWords(text,new_stops)
```

```
[1] "<>He</> went  bed       2 A.M. It's way  late!  He   20%   first,   eventually came."
```

Putting it All Together

We can combine some pre-processing steps into one call (there are, of course multiple ways to do this) – note that the order of implementation matters: a different order may very well lead to a different outcome.

```
tolower(
  tm::stripWhitespace(
    tm::removeWords(
      tm::removePunctuation(
        qdap::replace_symbol(
          qdap::replace_contraction(
            qdap::replace_abbreviation(
              qdap::bracketX(text)
            )
          )
        )
      )
      ,tm::stopwords("en"))
  )
)
```

```
[1] "he went bed 2 am way late he 20 percent asleep first sleep eventually came"
```

Word Stemming and Stem Completion

Stemming is also implemented in `tm`.

```
# Create sleep
(sleep <- c("sleepful","sleeps","sleeping"))

# Perform word stemming: stem_doc
(stem_doc <- tm::stemDocument(sleep))
```

```
[1] "sleepful" "sleeps"   "sleeping"
```

```
[1] "sleep" "sleep" "sleep"
```

```
# Create the completion dictionary: sleep_dict
sleep_dict <- c("sleep")

# Perform stem completion: complete_text
(complete_text <- tm::stemCompletion(stem_doc,sleep_dict))
```

```
  sleep   sleep   sleep
"sleep" "sleep" "sleep"
```

For illustrative purposes, let us take a quick look at a string with more substance.

```
text_data <- "In sleepful nights, Katia sleeps to achieve sleeping."
comp_dict <- c("In","sleep","nights","Katia","to","achieve")

# Remove punctuation
rm_punc <- tm::removePunctuation(text_data)

# Create character vector
n_char_vec <- unlist(strsplit(rm_punc, split = ' '))

# Perform word stemming: stem_doc
(stem_doc <- tm::stemDocument(n_char_vec))

# Re-complete stemmed document: complete_doc
(complete_doc <- tm::stemCompletion(stem_doc,comp_dict))
```

```
[1] "In"     "sleep"  "night"  "Katia"  "sleep"  "to"     "achiev" "sleep"
```

```
 In     sleep     night     Katia     sleep        to    achiev     sleep
"In"   "sleep"  "nights"   "Katia"   "sleep"      "to" "achieve"   "sleep"
```

Notice the slight difference between the stemmed string and the completed string.

Pre-Processing a Corpus

In practice, we never work with a single string or with a single document; how would we pre-process an entire corpus of text documents? The function `tm_map` maps the processing step to all documents in the corpus; if the processing function is not implemented in the package `tm`, it must be wrapped by the `content_transformer`.

As an example, consider the following customized pre-processing cleaner, which mixes base, `tm` and `qdap` functions.

```
clean_corpus <- function(corpus){
  corpus <- tm::tm_map(corpus, tm::content_transformer(qdap::replace_abbreviation))
  corpus <- tm::tm_map(corpus, tm::removePunctuation)
  corpus <- tm::tm_map(corpus, tm::removeNumbers)
  corpus <- tm::tm_map(corpus, tm::stemDocument)
  corpus <- tm::tm_map(corpus, tm::content_transformer(tolower))
  corpus <- tm::tm_map(corpus, tm::stripWhitespace)
  corpus <- tm::tm_map(corpus, tm::removeWords, c(tm::stopwords("en")))
  return(corpus)
}
```

We apply it to the Sens game recaps corpus `AP.recaps.corpus`.

```
clean_corp.AP.recaps <- clean_corpus(AP.recaps.corpus)
```

As an example, let's print the cleaned up recap for game 15 (compare with the raw text obtained previously).

```
clean_corp.AP.recaps[[15]][1]
```

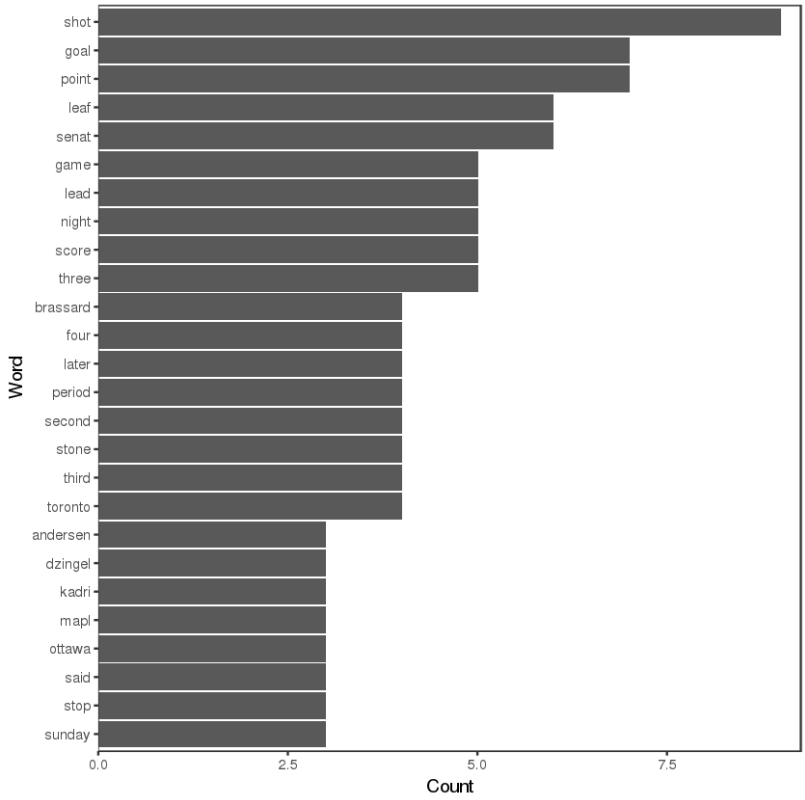
```
$content = 'team play  third game  four night  minnesota wild look plenti fresh  sunday night
  even  overtim matt dumba score late   extra session  darci kuemper stop shot help minnesota
  beat  ottawa senat  wild  come   loss  philadelphia  saturday  beat pittsburgh  thursday    end
  world  play backtoback   thought  held     good job wild coach bruce boudreau said ryan suter
  score  shorthand goal   first period  kuemper help  wild kill  three earli power play    team   [...]'
```

It is obviously not a proper English document, but the "meaning" can be gleamed fairly easily.[29] One important thing to keep in mind: **there is no secret pre-processing formula that will work with all corpora**. Context remains king/queen.

29: In no small part, due to the size of the document; with too few terms, it can be harder to make sense of the processed text.

We can revisit the first game recap we considered (game 56), and look at the new word counts. Originally, the eight most frequent terms ("the", "and", "a", "in", "for", "to", "on", "from") were English stopwords; what are the most frequent terms in the cleaned up corpus?

```
term_count <- freq_terms(clean_corp.AP.recaps[[56]][1],20)
plot(term_count)
```



Document-Term and Term-Document Matrices

The DTM and TDM can also be obtained from `tm`; we show how to create them from the clean game recaps corpus, starting with the DTM.

```
(AP.recaps_dtm <- tm::DocumentTermMatrix(clean_corp.AP.recaps))
```

```
<<DocumentTermMatrix (documents: 101, terms: 3293)>>
Non-/sparse entries: 22187/310406
Sparsity           : 93%
Maximal term length: 15
Weighting          : term frequency (tf)
```

Next, we convert `AP.recaps_dtm` to a matrix.

```
AP.recaps_m <- as.matrix(AP.recaps_dtm)
dim(AP.recaps_m)
```

```
[1]  101 3293
```

We can review a portion of the matrix (keep in mind that the default text statistic is the term frequency tf).

```
AP.recaps_m[79:84, 1005:1010]
    Terms
Docs ferland fewer fibula field fifth fifthround
  79       0     0      0     0     1          0
  80       0     0      0     0     0          0
  81       0     0      0     0     0          0
  82       0     0      0     0     0          0
  83       0     0      0     0     0          0
  84       0     0      0     0     0          0
```

We can do the same thing for the TDM.

```
(AP.recaps_tdm <- tm::TermDocumentMatrix(clean_corp.AP.recaps))
AP.recaps_m <- as.matrix(AP.recaps_tdm)
dim(AP.recaps_m)
AP.recaps_m[1005:1010, 79:84]
```

```
<<TermDocumentMatrix (terms: 3293, documents: 101)>>
Non-/sparse entries: 22187/310406
Sparsity           : 93%
Maximal term length: 15
Weighting          : term frequency (tf)

[1] 3293  101

            Docs
Terms        79 80 81 82 83 84
  ferland     0  0  0  0  0  0
  fewer       0  0  0  0  0  0
  fibula      0  0  0  0  0  0
  field       0  0  0  0  0  0
  fifth       1  0  0  0  0  0
  fifthround  0  0  0  0  0  0
```

Barchart of Frequent Terms with `tm`

These objects can be used to provide a BoW interpretation of the Senators' 2016-2017 season (regular season and playoffs).

We start by computing how often the terms appear in the entire corpus.

```
term_frequency <- rowSums(AP.recaps_m)
```

Next, we sort the term frequencies in descending order.

```
term_frequency <- sort(term_frequency, decreasing=TRUE)
```

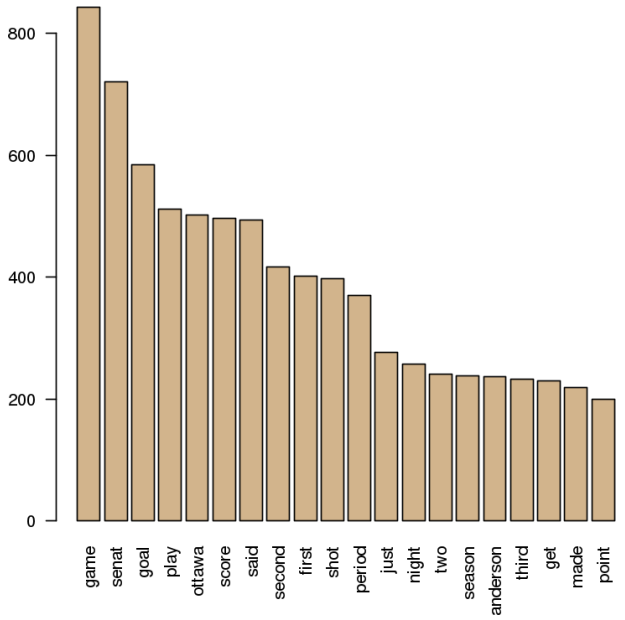
The top 20 most common words in the cleaned corpus are shown below (should we expect ties, as was the case when we looked at a single game recap?).

```
term_frequency[1:20]
```

```
game        score       period      anderson
    843          497          370          237
senat       said        just        third
    720          493          277          233
goal        second      night       get
    584          417          257          230
play        first       two         made
    512          402          240          218
ottawa      shot        season      point
    502          398          238          200
```

We can plot a barchart of the 20 most common words, or a word cloud of the (at most) 100 most common words.

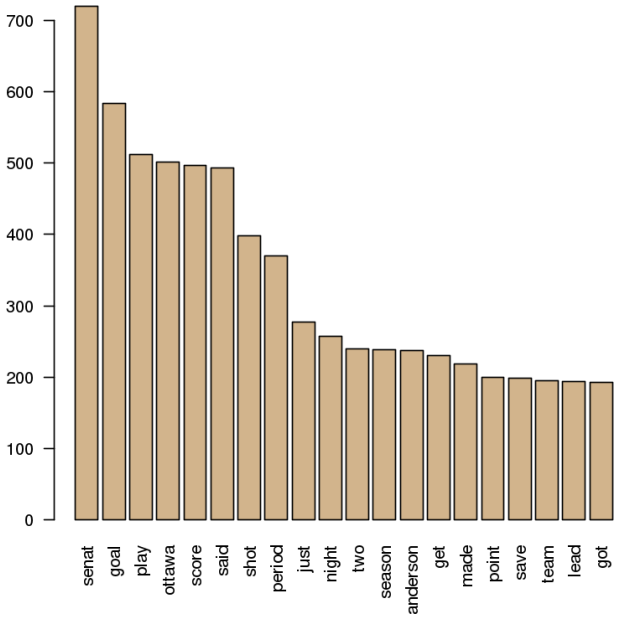
```
barplot(term_frequency[1:20], col = "tan", las = 2)

word_freqs = data.frame(term_frequency)
word_freqs$term = rownames(word_freqs)
word_freqs = word_freqs[,c(2,1)]
colnames(word_freqs)=c("term","num")

wordcloud::wordcloud(word_freqs$term, word_freqs$num, max.words=100, colors="red")
```

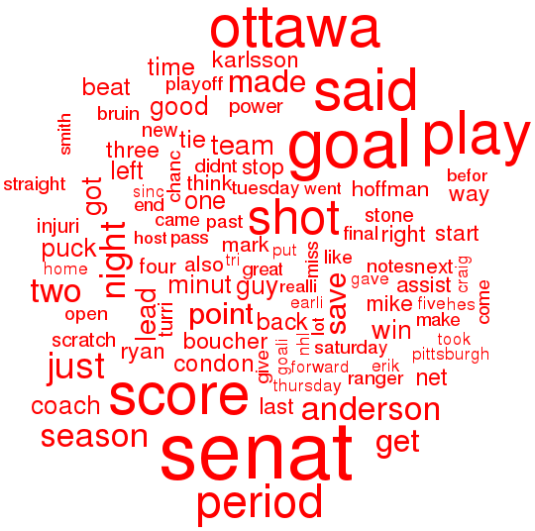


In practice, we already know that the corpus' documents are Ottawa Senators hockey game recaps, so we can remove frequent terms that do not add a lot of information due to the context.

```
clean_corpus_Sens <- function(corpus){
  corpus <- tm::tm_map(corpus, content_transformer(qdap::replace_abbreviation))
  corpus <- tm::tm_map(corpus, tm::removePunctuation)
  corpus <- tm::tm_map(corpus, tm::removeNumbers)
```

```
  corpus <- tm::tm_map(corpus, tm::stemDocument)
  corpus <- tm::tm_map(corpus, tm::content_transformer(tolower))
  corpus <- tm::tm_map(corpus, tm::stripWhitespace)
  corpus <- tm::tm_map(corpus, tm::removeWords, c(tm::stopwords("en"), "game", "first",
                       "second", "third", "Ottawa", "Senators"))
  return(corpus)
}

clean_corp2.AP.recaps <- clean_corpus_Sens(AP.recaps.corpus)
AP.recaps2_tdm <- tm::TermDocumentMatrix(clean_corp2.AP.recaps)
AP.recaps2_m <- as.matrix(AP.recaps2_tdm)

term_frequency2 <- rowSums(AP.recaps2_m)
term_frequency2 <- sort(term_frequency2, decreasing=TRUE)

barplot(term_frequency2[1:20], col = "tan", las = 2)

word_freqs2 = data.frame(term_frequency2)
word_freqs2$term = rownames(word_freqs2)
word_freqs2 = word_freqs2[,c(2,1)]
colnames(word_freqs2)=c("term","num")

wordcloud::wordcloud(word_freqs2$term, word_freqs2$num, max.words=100, colors="red")
```



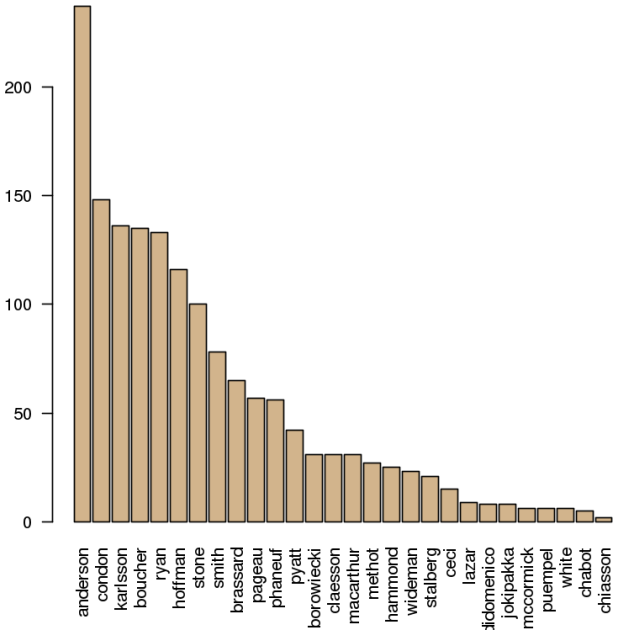


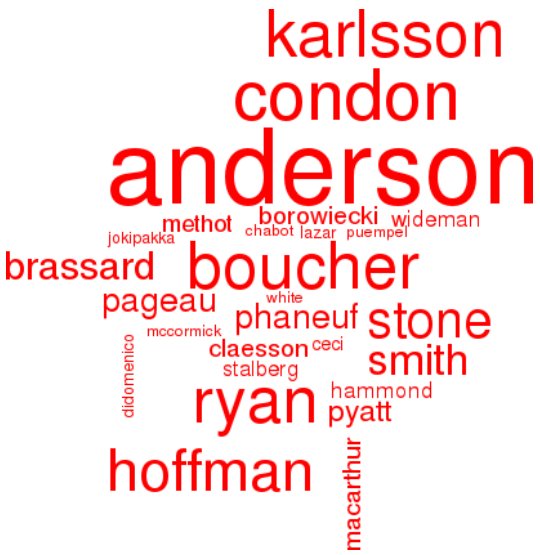
Do we get a better sense for how the season went? Assuming that you knew nothing about how things played out, would you be able to "predict" how close to winning the Stanley Cup the team came?

Finally, we will try to see if the recaps can help us determine the key players in the Senators' season.

```
keep=c("anderson","borowiecki","boucher","brassard","burrows","ceci","chabot","chiasson",
  "claesson","condon","didomenico","drieger","hammond","hoffman","jokipakka","karlsson",
  "lazar","macarthur","mccormick","methot","moore","pageau","phaneuf","puempel","pyatt",
  "ryan","ryans","smith","stalberg","stone","white","wideman","wingels")
```

```
word_freqs3 = word_freqs2[word_freqs2$term %in% keep, ]

barplot(term_frequency2[word_freqs2$term %in% keep], col = "tan", las = 2)
wordcloud::wordcloud(word_freqs3$term, word_freqs3$num, max.words=100, colors="red")
```





The beauty of the BoW approach is that even without any knowledge of the sport, it is rather straightforward to determine the players/personnel who were instrumental to the team's success that year.

27.4.2 Shakespeare vs. Marlowe

In this example, we introduce the basics of **tidy text mining** using the `tidytext` library in `R`, which shares syntax with H. Wickham's popular `tidyverse` suite of packages, which includes `ggplot2`, a powerful graphic library (see Chapter 1 and [9, ch. 13]).

Following [39], we will work with:

- a selection of Shakespeare's plays,
- a selection of Christopher Marlowe's play, and
- the Sens game recaps we work with in the preceding example.

The **tidytext** format (as do the other tidy formats) rely on the programming concept of **pipelines**.

The Pipeline Operator |>

(This section is a repeat of Section 1.4.1)

`R` is a **functional language**, which means that it uses nested parentheses, which can make code difficult to read. The **pipeline operator** `|>` (formerly `%>%`) and the `dplyr` package can be used to remedy the situation. Wickham[30] provided an example to illustrate how it works:

30: See [46] for everything there is to know about pipelines and tidy data.

```
hourly_delay <- filter(
   summarise(
     group_by(
       filter(
         flights,
         !is.na(dep_delay)
       ),
       date, hour
     ),
     delay = mean(dep_delay),
     n = n()
   ),
   n > 10
)
```

Without necessarily knowing how each of the internal functions works, we can still get a sense for what the overall nested structure does, and realize (albeit, with a fair amount of work) that the basic object on which we operate is the `flights` data frame.

The pipeline operator `|>` removes the need for nesting function calls, in favor of passing data from one function to the next:

```
library(dplyr)
hourly_delay <- flights |>
    filter(!is.na(dep_delay)) |>
    group_by(date, hour) |>
    summarise(delay = mean(dep_delay),n = n()) |>
    filter(n > 10)
```

It is now obvious that the flights data frame is the base object, for instance – the **gap** between pseudo-code and "code that runs" is significantly reduced. The beauty of this approach is that the block of code can now be 'read' directly: the `flights` data frame is

1. filtered (to remove missing values of the `dep_delay` variable);
2. grouped by `hours` within `days`;
3. the mean `delay` is calculated within `groups`, and
4. the mean `delay` is returned for those hours with more than `n > 10` flights.

The **pipeline rules** are simple – the object immediately to the left of the pipeline is passed as the first argument to the function immediately to its right:

- `data |> function` is equivalent to `function(data)`
- `data |> function(arg=value)` is equivalent to `function(data, arg=value)`

For instance:

```
library(dplyr)
swiss |> summary()
```

```
   Fertility      Agriculture     Examination      Education
 Min.   :35.00   Min.   : 1.20   Min.   : 3.00   Min.   : 1.00
 1st Qu.:64.70   1st Qu.:35.90   1st Qu.:12.00   1st Qu.: 6.00
 Median :70.40   Median :54.10   Median :16.00   Median : 8.00
 Mean   :70.14   Mean   :50.66   Mean   :16.49   Mean   :10.98
 3rd Qu.:78.45   3rd Qu.:67.65   3rd Qu.:22.00   3rd Qu.:12.00
 Max.   :92.50   Max.   :89.70   Max.   :37.00   Max.   :53.00

    Catholic       Infant.Mortality   threshold
 Min.   :  2.150   Min.   :10.80    Min.   :0.0000
 1st Qu.:  5.195   1st Qu.:18.15    1st Qu.:1.0000
 Median : 15.140   Median :20.00    Median :1.0000
 Mean   : 41.144   Mean   :19.94    Mean   :0.9362
 3rd Qu.: 93.125   3rd Qu.:21.70    3rd Qu.:1.0000
 Max.   :100.000   Max.   :26.60    Max.   :1.0000
```

The magrittr vignette provides additional information on the `magrittr` package, on which `dplyr` is based.

Tidy Text Structure

Tidy data has specific structure:[31]

31: See 1.4 for more information.

- each column represents a unique variable;
- each row represents a unique observation;
- each table represents a unique type of observational unit.

Tidy text is a table with one **token** (single word, $n-$gram, sentence, paragraph) per row, assuming that words have been tokenized to commonly-used **units of text**.

As an example, consider the following haiku by master Matsuo Basho.

```
haiku <- c('In the twilight rain',
           'these brilliant-hued hibiscus -',
           'A lovely sunset')
haiku
```

```
[1] "In the twilight rain"   "these brilliant-hued hibiscus -"
[3] "A lovely sunset"
```

We turn it into a data frame.

```
(haiku.df <- data.frame(text=haiku,
                        stringsAsFactors = FALSE))
```

```
                           text
1            In the twilight rain
2 these brilliant-hued hibiscus -
3                 A lovely sunset
```

In the `data.frame()` call above, the last parameter is important as we want to be able to separate the text into constituents tokens (words). We can **unnest** the tokens of the haiku as follows.

```
library(tidytext)
haiku.df |> unnest_tokens(word,text)
```

```
       word
1        in
2       the
3   twilight
4      rain
5     these
6  brilliant
7      hued
8   hibiscus
9         a
10    lovely
11    sunset
```

The `tidytext` function `unnest_token()` separates the tokens (words, in this example), strips away the punctuation, and converts to lowercase.

Tidy Text Flow

In the tidy text framework, we generally:

1. start with text data;
2. unnest the tokens to produce the first iteration of tidy text;
3. clean the tidy text as required;
4. summarize the tidy text into a first iteration of summarized text;
5. clean and analyze the summarized text, and
6. visualize and present the text mining results.

Tidy Text Analysis

We illustrate the flow with the help of some of Shakespeare's plays, available at the Gutenberg Project (Project ID – *Romeo and Juliet*: 1112; *Hamlet*: 1524; *Macbeth*: 2264; *A Midsummer Night's Dream*: 2242, etc.).

```
library(gutenbergr)
will_shakespeare <- gutenberg_download(c(1790,2240,2242,
                          2243,2246,2250,2251,2253,2262,
                          2264,2267,2268,23042,23046))
head(will_shakespeare,20)
```

```
# A tibble: 20 x 2
   gutenberg_id text
          <int> <chr>
 1         1790 "**********************************************************************"
 2         1790 "THIS EBOOK WAS ONE OF PROJECT GUTENBERG'S EARLY FILES PRODUCED AT A"
 3         1790 "TIME WHEN PROOFING METHODS AND TOOLS WERE NOT WELL DEVELOPED. THERE"
 4         1790 "IS AN IMPROVED EDITION OF THIS TITLE WHICH MAY BE VIEWED AS EBOOK"
 5         1790 "(#1528) at https://www.gutenberg.org/ebooks/1528"
 6         1790 "**********************************************************************"
```

```
 7         1790 ""
 8         1790 ""
 9         1790 "Troilus and Cressida, World Library edition, several typos fixed."
10         1790 ""
11         1790 "This Etext file is presented by Project Gutenberg, in"
12         1790 "cooperation with World Library, Inc., from their Library of the"
13         1790 "Future and Shakespeare CDROMS.  Project Gutenberg often releases"
14         1790 "Etexts that are NOT placed in the Public Domain!!"
15         1790 ""
16         1790 "*This Etext has certain copyright implications you should read!*"
17         1790 ""
18         1790 "<<THIS ELECTRONIC VERSION OF THE COMPLETE WORKS OF WILLIAM"
19         1790 "SHAKESPEARE IS COPYRIGHT 1990-1993 BY WORLD LIBRARY, INC., AND IS"
20         1790 "PROVIDED BY PROJECT GUTENBERG WITH PERMISSION.  ELECTRONIC AND"
```

Not the most stirring literature, to be sure. We now produce (and clean) the corresponding tidy text dataset.

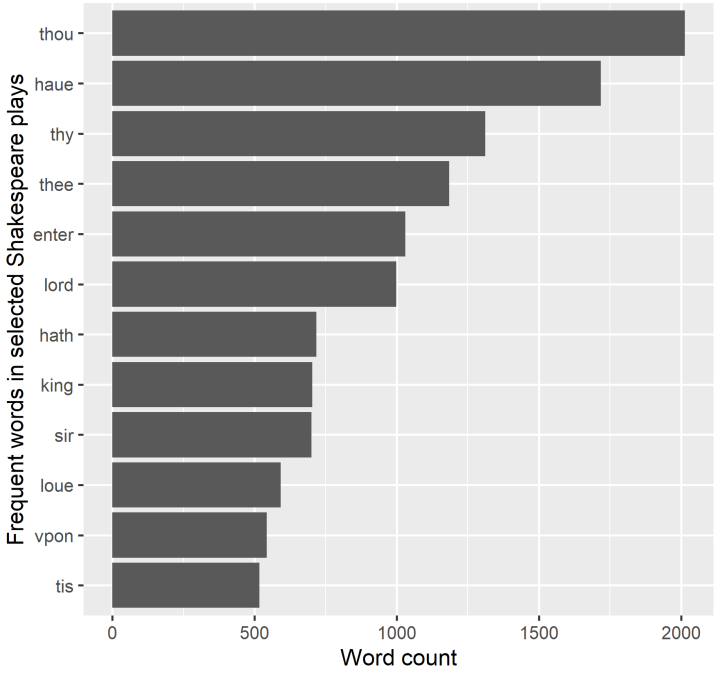
```
library(stringr) # necessary to use str_extract

tidy_ws <- will_shakespeare |>
  unnest_tokens(word,text) |>
  dplyr::mutate(word = str_extract(word,"[a-z']+")) |> # removing stray punctuation, etc.
  dplyr::anti_join(stop_words) |>  # removing the heading business
  na.omit() # removing NAs

head(tidy_ws)
```

```
# A tibble: 6 x 2
  gutenberg_id word
         <int> <chr>
1         1790 ebook
2         1790 project
3         1790 gutenberg's
4         1790 files
5         1790 produced
6         1790 time
```

We can easily produce a word count for this data frame.

```
library(ggplot2)
tidy_ws |> dplyr::count(word, sort=TRUE)

tidy_ws |> dplyr::count(word, sort=TRUE) |>
  dplyr::filter(n > 500) |>
  dplyr::mutate(word=reorder(word,n)) |>
  ggplot(aes(word,n)) +
    geom_col() +
    xlab("Frequent words in selected Shakespeare plays") +
    ylab("Word count") +
    coord_flip()
```

```
# A tibble: 20,297 x 2
   word      n
   <chr> <int>
 1 thou   2014
 2 haue   1718
 3 thy    1311
 4 thee   1185
 5 enter  1030
 6 lord    999
 7 hath    717
 8 king    703
 9 sir     700
10 loue    593
# ... with 20,287 more rows
```



Shakespeare and Marlowe

We can do the same for Christopher Marlowe, a contemporary of Shakespeare.

```
kit_marlowe <-gutenberg_download(c(901,1094,1496,1589,
                                   16169,18781,20288))

tidy_km <- kit_marlowe |>  unnest_tokens(word,text) |>
  dplyr::mutate(word = str_extract(word,"[a-z']+")) |>
  dplyr::anti_join(stop_words) |>
  na.omit() # remove NAs

head(tidy_km)
```

```
# A tibble: 6 x 2
  gutenberg_id word
         <int> <chr>
1          901 jew
2          901 malta
3          901 christopher
4          901 marlowe
5          901 edited
6          901 rev
```

Next, we look at both of these datasets **simultaneously**. In order to do so, we build a word count data set with the help of the pipeline operator. One of its advantages is that we can build the query sequentially and easily see the output at various stages.

We start by binding `tidy_ws` and `tidy_km` into a single data frame.

```
word_count <- dplyr::bind_rows(dplyr::mutate(tidy_ws,author="WillShakespeare"),
                               dplyr::mutate(tidy_km,author="KitMarlowe"))
head(word_count)
tail(word_count)
```

```
# A tibble: 6 x 3
  gutenberg_id word        author
         <int> <chr>       <chr>
1         1790 ebook       WillShakespeare
2         1790 project     WillShakespeare
3         1790 gutenberg's WillShakespeare
4         1790 files       WillShakespeare
5         1790 produced    WillShakespeare
6         1790 time        WillShakespeare

# A tibble: 6 x 3
  gutenberg_id word      author
         <int> <chr>     <chr>
1        20288 mine      KitMarlowe
2        20288 eyes      KitMarlowe
3        20288 witness   KitMarlowe
4        20288 grief     KitMarlowe
5        20288 innocency KitMarlowe
6        20288 exeunt    KitMarlowe
```

Next, we execute a word count for each of the authors (note the sorting of the outputs, and the new field `n`).

```
word_count <- dplyr::bind_rows(dplyr::mutate(tidy_ws,author="WillShakespeare"),
                               dplyr::mutate(tidy_km,author="KitMarlowe")) |>
              dplyr::count(author,word) |>  dplyr::group_by(author)

head(word_count)
tail(word_count)
```

```
# A tibble: 6 x 3
# Groups:   author [1]
  author     word          n
  <chr>      <chr>     <int>
1 KitMarlowe abandon       3
2 KitMarlowe abandon'd     1
3 KitMarlowe abandons      1
4 KitMarlowe abate         2
5 KitMarlowe abated        1
6 KitMarlowe abb           1

# A tibble: 6 x 3
# Groups:   author [1]
  author          word        n
  <chr>           <chr>   <int>
1 WillShakespeare zeal        2
2 WillShakespeare zeale       9
3 WillShakespeare zeales      1
4 WillShakespeare zealous     1
5 WillShakespeare zenith      1
6 WillShakespeare zip         1
```

In order to use the tidy approach, we need `word_count` to have a unique value for each word for each author. Note that the size of each of the author datasets is different, as we are using a higher number of Shakespeare plays. Rather than look at **raw counts** (which would naturally favour the Bard's output), we consider proportions:

$$\frac{\text{number of occurrences of a specific term in an author's dataset}}{\text{total number of terms in an author's dataset}}.$$

```
word_count <- dplyr::bind_rows(dplyr::mutate(tidy_ws,author="WillShakespeare"),
                               dplyr::mutate(tidy_km,author="KitMarlowe")) |>
              dplyr::count(author,word) |>  dplyr::group_by(author) |>
              dplyr::mutate(proportion = n / sum(n))

head(word_count)
tail(word_count)
```

```
# A tibble: 6 x 4
# Groups:   author [1]
  author     word          n proportion
  <chr>      <chr>     <int>      <dbl>
1 KitMarlowe abandon       3  0.0000513
2 KitMarlowe abandon'd     1  0.0000171
3 KitMarlowe abandons      1  0.0000171
4 KitMarlowe abate         2  0.0000342
5 KitMarlowe abated        1  0.0000171
6 KitMarlowe abb           1  0.0000171

# A tibble: 6 x 4
# Groups:   author [1]
  author          word        n proportion
```

```
  <chr>           <chr>   <int>      <dbl>
1 WillShakespeare zeal        2 0.0000136
2 WillShakespeare zeale       9 0.0000611
3 WillShakespeare zeales      1 0.00000679
4 WillShakespeare zealous     1 0.00000679
5 WillShakespeare zenith      1 0.00000679
6 WillShakespeare zip         1 0.00000679
```

We can now remove the raw counts and focus solely on the proportions.

```
word_count <- dplyr::bind_rows(dplyr::mutate(tidy_ws,author="WillShakespeare"),
                               dplyr::mutate(tidy_km,author="KitMarlowe")) |>
              dplyr::count(author,word) |>  dplyr::group_by(author) |>
              dplyr::mutate(proportion = n / sum(n)) |> dplyr::select(-c(n))

word_count
```

```
# A tibble: 31,489 x 3
# Groups:   author [2]
      author              word    proportion
      <chr>               <chr>        <dbl>
    1 KitMarlowe       abandon    0.0000513
    2 KitMarlowe       abandon'd  0.0000171
    3 KitMarlowe       abandons   0.0000171
    4 KitMarlowe       abate      0.0000342
    5 KitMarlowe       abated     0.0000171
    6 KitMarlowe       abb        0.0000171
      ...
31484 WillShakespeare  zeal       0.0000136
31485 WillShakespeare  zeale      0.0000611
31486 WillShakespeare  zeales     0.00000679
31487 WillShakespeare  zealous    0.00000679
31488 WillShakespeare  zenith     0.00000679
31489 WillShakespeare  zip        0.00000679
```

Next, we reshape `word_count` to facilitate the analysis: each word is now represented by a row, and the proportion of the time it appears in each author's writings is shown in the corresponding column.

```
word_count <- dplyr::bind_rows(dplyr::mutate(tidy_ws,author="WillShakespeare"),
                               dplyr::mutate(tidy_km,author="KitMarlowe")) |>
              dplyr::count(author,word) |>  dplyr::group_by(author) |>
              dplyr::mutate(proportion = n / sum(n)) |> dplyr::select(-c(n)) |>
              tidyr::spread(author,proportion)

word_count
```

```
# A tibble: 25,020 x 3
       word   KitMarlowe WillShakespeare
       <chr>       <dbl>           <dbl>
 1 a'kin      NA              0.00000679
 2 a'th       NA              0.0000814
```

```
 3 a'that     NA              0.00000679
 4 abandon     0.0000513      0.0000136
 5 abandon'd   0.0000171      0.0000136
 6 abandons    0.0000171     NA
 7 abash'd    NA              0.00000679
 8 abate       0.0000342      0.0000475
 9 abated      0.0000171     NA
10 abates     NA              0.00000679
# ... with 25,010 more rows
```

We can easily see what proportion of each author's output is not found in the other's.

```
# % of Shakespeare's output terms not in Marlowe's
(WS_nKM <- sum(word_count$WillShakespeare[is.na(word_count$KitMarlowe)]))
# % of Marlowe's output terms not in Shakespeare's
(KM_nWS <- sum(word_count$KitMarlowe[is.na(word_count$WillShakespeare)]))
```

```
[1] 0.3092499
[1] 0.2497649
```

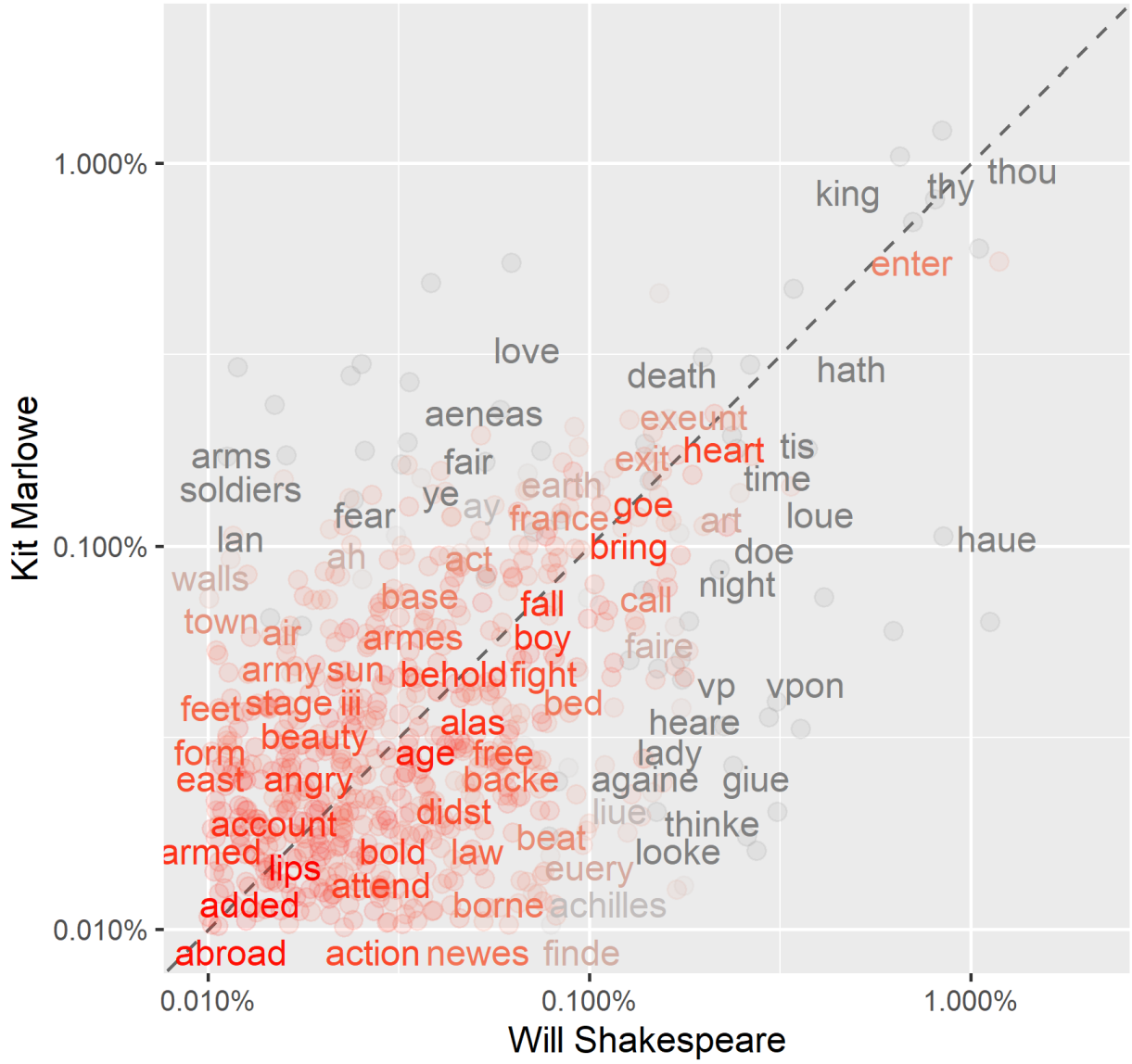
Do these proportions seem high, given that they were contemporaries? Finally, we re-organize the table for use with `ggplot()` (strictly-speaking, this step is not mandatory, but the charts we produce will look nicer).

```
word_count <- dplyr::bind_rows(dplyr::mutate(tidy_ws,author="WillShakespeare"),
                               dplyr::mutate(tidy_km,author="KitMarlowe")) |>
              dplyr::count(author,word) |>  dplyr::group_by(author) |>
              dplyr::mutate(proportion = n / sum(n)) |> dplyr::select(-c(n)) |>
              tidyr::spread(author,proportion) |>
              tidyr::gather(author, proportion, 'WillShakespeare')
```

Here is a logarithmic scale scatterplot of word usage by both authors (for words that were used by both).

```
library(scales)
ggplot(word_count, aes(x = proportion, y = 'KitMarlowe',
                       color = abs('KitMarlowe' - proportion))) +
  geom_abline(color = "gray40", lty = 2) +
  geom_jitter(alpha = 0.1, size = 2.5, width = 0.3, height = 0.3) +
  geom_text(aes(label = word), check_overlap = TRUE, vjust = 1.5) +
  scale_x_log10(labels = percent_format(),limits=c(0.0001,0.02)) +
  scale_y_log10(labels = percent_format(),limits=c(0.0001,0.02)) +
  scale_color_gradient(limits = c(0, 0.001), low = "red", high = "gray75") +
  theme(legend.position="none") +
  labs(y = "Kit Marlowe", x = "Will Shakespeare")
```

```
Warning messages:
1: Removed 24363 rows containing missing values
(geom_point).
2: Removed 24205 rows containing missing values
(geom_text).
```



Words near the straight line are used with roughly the same frequency by both authors. For instance: "king", "thou", and "thy" in the high-frequency spectrum, and "angry", "alas", and "behold" in the low-frequency spectrum.

Words away from the straight line are used more frequently by one of the authors: "lady" and "achilles" seem to be used relatively more often by Shakespeare than by Marlowe, while "aeneas" is in the opposite situation (these terms are specific to plays).

The colour is related to the (real) distance between the relative frequencies of a term for each author (red is near, gray is far) – the logarithmic scales of both axes explain the shape of the red cloud (large at the bottom, thin at the top).[32]

Note the presence of "armes", "arms", and "armed", or of "love" and "loue" – what does that tell us about the text (and the English used)? Should we be surprised about the prevalence of terms like "enter", "exit", and "exeunt"?

32: Do you see why this needs to be the case?

Finally, let's see if we can quantify the similarity in word usage.

```
cor.test(data = word_count, ~ proportion + 'KitMarlowe')
```

```
  Pearson's product-moment correlation

data:  proportion and KitMarlowe
t = 89.335, df = 6467, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.7321162 0.7539396
sample estimates:
      cor
0.7432256
```

There's a fairly strong correlation (0.74) between the relative term frequencies for the two wordsmiths (among those terms which are found in both text outputs – recall `KM_nWS` and `WS_nKM`). That should not be entirely unexpected, since they were contemporaries: one would naïvely predict that the depth of their vocabulary and the way they deployed it would be linked, to some extent.

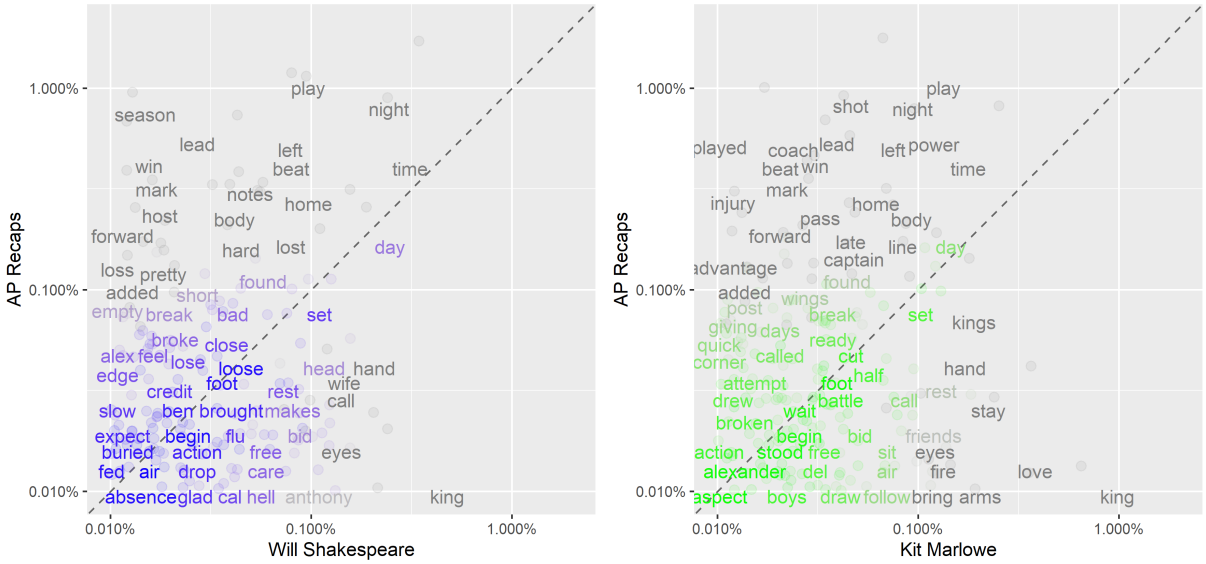
But without comparisons to other texts, it is difficult to really put this value in perspective.

Shakespeare and Sens Game Recaps

Let's see how Shakespeare and Marlowe compare to a modern body of work, the NHL Senators' game recaps from the previous section.

```
recaps <- read.csv(file="Recap_data.csv", header=TRUE, sep=",", stringsAsFactors=FALSE)

AP.recaps <- recaps$AP_Recap

recaps.df <- data.frame(text=AP.recaps, stringsAsFactors = FALSE)

tidy_AP <- recaps.df |>
  tidytext::unnest_tokens(word,text) |>
  dplyr::mutate(word = str_extract(word,"[a-z']+")) |>
  dplyr::anti_join(stop_words) |>
  na.omit() # remove NAs

head(tidy_AP) # inspect

word_count_2 <- dplyr::bind_rows(dplyr::mutate(tidy_ws,author="WillShakespeare"),
                                 dplyr::mutate(tidy_AP,author="AP_recaps")) |>
                dplyr::count(author,word) |>
                dplyr::group_by(author) |>
                dplyr::mutate(proportion = n / sum(n)) |>
                dplyr::select(-c(n)) |>
                tidyr::spread(author,proportion) |>
                tidyr::gather(author, proportion, 'WillShakespeare')
```

```
word_count_3 <- dplyr::bind_rows(dplyr::mutate(tidy_km,author="KitMarlowe"),
                                 dplyr::mutate(tidy_AP,author="AP_recaps")) |>
                dplyr::count(author,word) |>
                dplyr::group_by(author) |>
                dplyr::mutate(proportion = n / sum(n)) |>
                dplyr::select(-c(n)) |>
                tidyr::spread(author,proportion) |>
                tidyr::gather(author, proportion, `KitMarlowe`)

ggplot(word_count_2, aes(x = proportion, y = `AP_recaps`,
                         color = abs(`AP_recaps` - proportion))) +
  geom_abline(color = "gray40", lty = 2) +
  geom_jitter(alpha = 0.1, size = 2.5, width = 0.3, height = 0.3) +
  geom_text(aes(label = word), check_overlap = TRUE, vjust = 1.5) +
  scale_x_log10(labels = percent_format(),limits=c(0.0001,0.02)) +
  scale_y_log10(labels = percent_format(),limits=c(0.0001,0.02)) +
  scale_color_gradient(limits = c(0, 0.001), low = "blue", high = "gray75") +
  theme(legend.position="none") +
  labs(y = "AP Recaps", x = "Will Shakespeare")

ggplot(word_count_3, aes(x = proportion, y = `AP_recaps`,
                         color = abs(`AP_recaps` - proportion))) +
  geom_abline(color = "gray40", lty = 2) +
  geom_jitter(alpha = 0.1, size = 2.5, width = 0.3, height = 0.3) +
  geom_text(aes(label = word), check_overlap = TRUE, vjust = 1.5) +
  scale_x_log10(labels = percent_format(),limits=c(0.0001,0.02)) +
  scale_y_log10(labels = percent_format(),limits=c(0.0001,0.02)) +
  scale_color_gradient(limits = c(0, 0.001), low = "green", high = "gray75") +
  theme(legend.position="none") +
  labs(y = "AP Recaps", x = "Kit Marlowe")
```



We can see the proportion of terms not found in the other corpora.

```
# % of Shakespeare's terms not in the game recaps
(WS_nAP <- sum(word_count_2$proportion[is.na(word_count_2$AP_recaps)]))
```

```
# % of game recaps' terms not in the selected Shakespeare plays
(AP_nWS <- sum(word_count_2$AP_recaps[is.na(word_count_2$proportion)]))
# % of Shakespeare's terms not in the game recaps
(KM_nAP <- sum(word_count_3$proportion[is.na(word_count_3$AP_recaps)]))
# % of game recaps' terms not in the selected Marlowe plays
(AP_nKM <- sum(word_count_3$AP_recaps[is.na(word_count_3$proportion)]))
```

```
[1] 0.8026031
[1] 0.4804179
[1] 0.7732845
[1] 0.5588203
```

The proportions are much higher than when comparing Marlowe's and Shakespeare's outputs. In light of the differences in terms of topics, style, and linguistic drift over the centuries, should any of these be surprising?

The corresponding correlations are shown below.

```
cor.test(data = word_count_2, ~ proportion + `AP_recaps`)
cor.test(data = word_count_3, ~ proportion + `AP_recaps`)
```

```
  Pearson's product-moment correlation

data:  proportion and AP_recaps
t = 2.8059, df = 1377, p-value = 0.005089
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.02270094 0.12767835
sample estimates:
       cor
0.07539856

  Pearson's product-moment correlation

data:  proportion and AP_recaps
t = 1.2599, df = 1142, p-value = 0.208
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.02074664  0.09501030
sample estimates:
       cor
0.03725681
```

Again, it should come as no surprise that the recap writers do not use English in the same BoW way that Shakespeare and Marlowe did.

It would be interesting to see if these results are stable under a different subset of Shakespeare and Marlowe plays.

$n-$Grams

Up to this point, we have been using **word**, **term**, **token**, **unit** interchangeably when analyzing text, as befits the BoW approach.

It's not too difficult to think of applications where the basic numerical unit is not the relative frequency (or tf-idf) of **single words**, however, but the **links between 2 or more words**, in succession or in co-occurrence. Rather than tokenize some text by words, we can tokenize it by series of n **consecutive words** (also called $n-$**grams**).

In what follows, we focus on $n = 2$. Are there interesting **bigrams** in Shakespeare's plays? What would we expect his common bigrams to be?

```
tidy_ws.2 <- will_shakespeare |>
  tidytext::unnest_tokens(bigram,text,token="ngrams",n=2) |>
  dplyr::mutate(bigram = stringr::str_extract(bigram,"[0-9a-zA-Z'\ ]+")) |>
  dplyr::count(bigram,sort=TRUE) |>
  na.omit() # produce a count and sort on decreasing frequency

tidy_ws.2
```

```
# A tibble: 143,344 x 2
   bigram      n
   <chr>   <int>
 1 i am      671
 2 in the    636
 3 my lord   604
 4 i will    584
 5 of the    581
 6 to the    528
 7 i haue    517
 8 it is     424
 9 that i    319
10 and the   305
11 to be     304
12 and i     287
13 is the    284
14 i would   256
15 of my     253
16 i know    244
17 i do      240
18 you are   234
19 if you    222
20 is not    220
# ... with 143,324 more rows
```

There are a **lot more** bigrams than there were individual terms, which makes sense from a combinatorial perspective. At first glance, among the top 10 most frequent bigrams, only one conveys even a sliver of information: "my lord". Everything else is stopword material.

However, what about the 11th most frequent bigram? In a general context, "to be" is a stopword bigram – but there is at least a few specific instance

in the Shakespearean context where that specific bigram is emphatically not just a "stopword".[33]

Removing bigram stopwords is simple, although not as straigthforward as in the unigram case:

1. split the two members of the bigrams into 2 columns;
2. verify if each, separately, is a regular stopword, and
3. remove the bigrams for which one of the components is a stopword.

For the sake of this exercise, let's also remove words related to the printing business, and theatre terms.

33: "To be, or not to be, that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take Arms against a Sea of troubles,
And by opposing end them: to die, to sleep
No more; and by a sleep, to say we end
The heart-ache, and the thousand natural shocks
That Flesh is heir to?
– *Hamlet* (Act 3, Scene 1)

```
word = c("gutenberg","shakespeare","","etext","1990","1993","public","print","copies"
        ,"membership","commercial","commercially","electronic","download","distribution"
        ,"ff","f1","f2","f3","f4","NA","collier","ms","cap","txt","zip"
        ,"library","printed", "text","editions"
        ,"executive", "pobox", "fees", "million", "ascii", "legal", "61825", "2782"
        ,"director", "machine","readable","carnegie","mellon","university"
        ,"exit", "exeunt", "enter", "scene", "act", "folio", "dramatis"
        ,"mine","tis", "thine","thy", "thou","art","hast", "shalt","dost","thee"
        ,"act_4","act_1","act_2","act_3","act_5","sc_1","sc_2","sc_3","sc_4","sc_5"
        ,"sc_6","sc_7","sc_8","sc_9","sc_10","sc_11")
lexicon = rep("modern",length(word)) # let's call it the modern lexicon
addition = data.frame(word,lexicon)
stop_words_ws = rbind(stop_words,addition)

tidy_ws.2_cleaned <- tidy_ws.2 |>
  tidyr::separate(bigram, c("FirstTerm","SecondTerm"), sep=" ") |>
  dplyr::filter(!FirstTerm %in% stop_words_ws$word) |>
  dplyr::filter(!SecondTerm %in% stop_words_ws$word)

tidy_ws.2_cleaned <- tidy_ws.2_cleaned[!is.na(tidy_ws.2_cleaned$FirstTerm) &
                                       !is.na(tidy_ws.2_cleaned$SecondTerm), ]
tidy_ws.2_cleaned <- tidy_ws.2_cleaned |>
                     tidyr::unite(bigram,FirstTerm,SecondTerm, sep=" ")
tidy_ws.2_cleaned
```

```
# A tibble: 31,721 x 2
   bigram         n
   <chr>      <int>
 1 haue beene    32
 2 sir iohn      30
 3 ha ha         27
 4 om pope       27
 5 anon conj     23
 6 haue heard    23
 7 haue lost     22
 8 haue seene    22
 9 noble lord    21
10 hath beene    17
# ... with 31,711 more rows
```

Other bigram and $n-$gram ideas can be found in Section 27.5.

27.4.3 The Play's the Thing

In this section, we will take a more in-depth look at **text visualizations**, which play a role just as important in text analysis as visualizations do in numerical data science. We are somewhat hampered by the lack of numerical values, but there are **workarounds**.

We will work with a set of Shakespearean plays, categorized into **comedies**, **tragedies**, and **histories**. We will use the `tm` and `qdap` libraries in `R`, among others.

Loading the Data

We start by loading the data into three corpora.

```
corpus_C <- tm::Corpus(tm::DirSource("ShakespeareComedies/"),
                       readerControl=list(language="lat"))
corpus_T <- tm::Corpus(tm::DirSource("ShakespeareTragedies/"),
                       readerControl=list(language="lat"))
corpus_H <- tm::Corpus(tm::DirSource("ShakespeareHistories/"),
                       readerControl=list(language="lat"))
summary(corpus_C)
summary(corpus_T)
summary(corpus_H)
```

```
# corpus C
                                       Length Class             Mode
A_Midsummer_Nights_Dream_first_pass.txt   2      PlainTextDocument list
Alls_Well_That_Ends_Well_first_pass.txt   2      PlainTextDocument list
As_You_Like_It_first_pass.txt             2      PlainTextDocument list
Cymbeline_first_pass.txt                  2      PlainTextDocument list
Loves_Labours_Lost_first_pass.txt         2      PlainTextDocument list
Measure_for_Measure_first_pass.txt        2      PlainTextDocument list
Much_Ado_About_Nothing_first_pass.txt     2      PlainTextDocument list
Pericles_Prince_of_Tyre_first_pass.txt    2      PlainTextDocument list
Taming_of_the_Shrew_first_pass.txt        2      PlainTextDocument list
The_Comedy_of_Errors_first_pass.txt       2      PlainTextDocument list
The_Merchant_of_Venice_first_pass.txt     2      PlainTextDocument list
The_Merry_Wives_of_Windsor_first_pass.txt 2      PlainTextDocument list
The_Tempest_first_pass.txt                2      PlainTextDocument list
Troilus_and_Cressida_first_pass.txt       2      PlainTextDocument list
Twelfth_Night_first_pass.txt              2      PlainTextDocument list
Two_Gentlemen_of_Verona_first_pass.txt    2      PlainTextDocument list
Winters_Tale_first_pass.txt               2      PlainTextDocument list

# corpus_T
                                    Length Class             Mode
Antony_and_Cleopatra_first_pass.txt 2      PlainTextDocument list
Coriolanus_first_pass.txt           2      PlainTextDocument list
Hamlet_first_pass.txt               2      PlainTextDocument list
Julius_Caesar_first_pass.txt        2      PlainTextDocument list
King_Lear_second_pass.txt           2      PlainTextDocument list
Macbeth_first_pass.txt              2      PlainTextDocument list
```

```
Othello_first_pass.txt              2      PlainTextDocument list
Romeo_and_Juliet_first_pass.txt     2      PlainTextDocument list
Timon_of_Athens_first_pass.txt      2      PlainTextDocument list
Titus_Andronicus_first_pass.txt     2      PlainTextDocument list

# corpus_H
                                  Length Class             Mode
Henry_IV_part_1_first_pass.txt    2      PlainTextDocument list
Henry_IV_part_2_first_pass.txt    2      PlainTextDocument list
Henry_V_first_pass.txt            2      PlainTextDocument list
Henry_VI_part_1_first_pass.txt    2      PlainTextDocument list
Henry_VI_part_2_first_pass.txt    2      PlainTextDocument list
Henry_VI_part_3_first_pass.txt    2      PlainTextDocument list
Henry_VIII_first_pass.txt         2      PlainTextDocument list
King_John_first_pass.txt          2      PlainTextDocument list
Richard_II_first_pass.txt         2      PlainTextDocument list
Richard_III_first_pass.txt        2      PlainTextDocument list
```

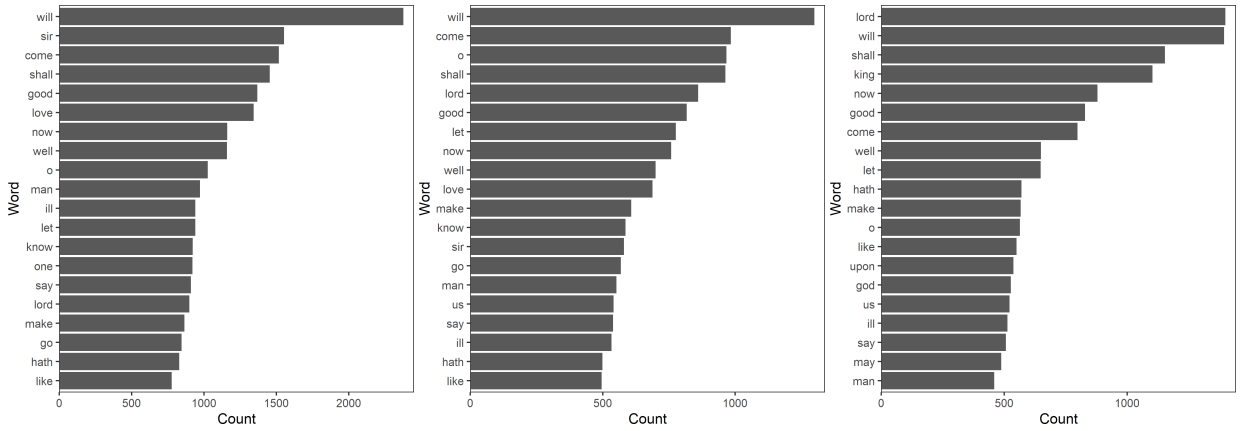
Cleaning the Data

Next, we build a cleaning function for the text and apply it to each corpus.

```
clean_corpus <- function(corpus){
  corpus <- tm::tm_map(corpus, tm::removePunctuation)
  corpus <- tm::tm_map(corpus, tm::removeNumbers)
  corpus <- tm::tm_map(corpus, tm::stemDocument, language="english")
  corpus <- tm::tm_map(corpus, tm::content_transformer(tolower))
  corpus <- tm::tm_map(corpus, tm::stripWhitespace)
  corpus <- tm::tm_map(corpus, tm::removeWords,
                       c(tm::stopwords("english"),
                       c("I", "and", "the", "that", "thou", "thee", "thi", "â")))
  return(corpus)
}

clean_C = clean_corpus(corpus_C)
clean_T = clean_corpus(corpus_T)
clean_H = clean_corpus(corpus_H)
```

We find the 20 most frequent terms in each corpus.

```
term_count_C <- qdap::freq_terms(clean_C,20)
term_count_T <- qdap::freq_terms(clean_T,20)
term_count_H <- qdap::freq_terms(clean_H,20)

plot(term_count_C)
plot(term_count_T)
plot(term_count_H)
```

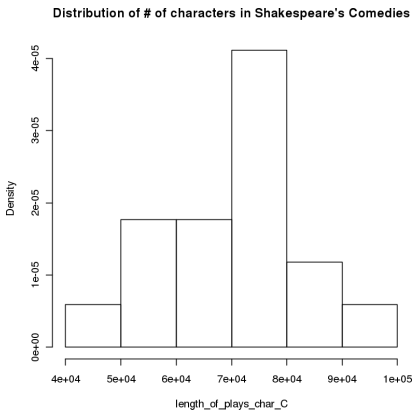


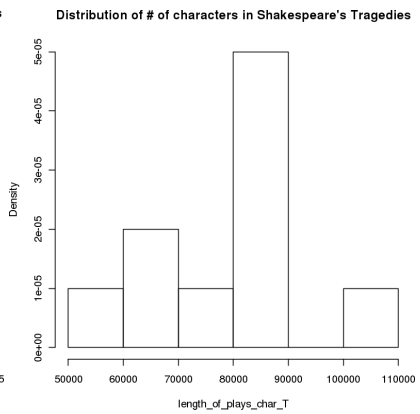
comedies tragedies histories

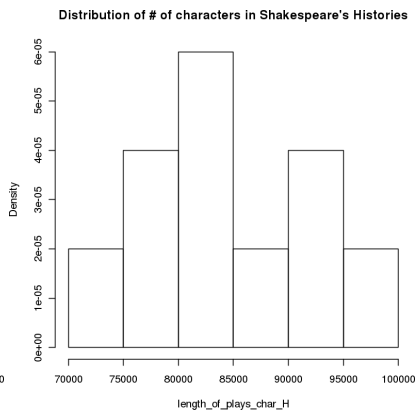
Basic Statistics

We can also take a look at some basic statistics regarding the number of characters (letters, not people) and the number of words in each play.

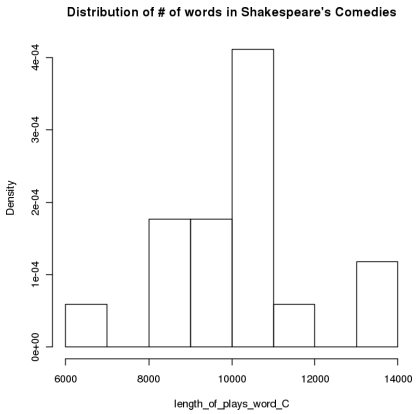
```
length_of_plays_char_C <- vector(mode="numeric", length=17)
for(j in 1:17){length_of_plays_char_C[j]=nchar(clean_C[[j]][1])}
hist(length_of_plays_char_C, freq=F, main="Distribution of # of char in Shakespeare's Comedies")
summary(length_of_plays_char_C)

length_of_plays_char_T <- vector(mode="numeric", length=10)
for(j in 1:10){length_of_plays_char_T[j]=nchar(clean_T[[j]][1])}
hist(length_of_plays_char_T, freq=F, main="Distribution of # of char in Shakespeare's Tragedies")
summary(length_of_plays_char_T)

length_of_plays_char_H <- vector(mode="numeric", length=10)
for(j in 1:10){length_of_plays_char_H[j]=nchar(clean_H[[j]][1])}
hist(length_of_plays_char_H, freq=F, main="Distribution of # of char in Shakespeare's Histories")
summary(length_of_plays_char_H)

length_of_plays_word_C <- vector(mode="numeric", length=17)
for(j in 1:17){length_of_plays_word_C[j]=length(
               strsplit(gsub(' {2,}',' ',clean_C[[j]][1]),' ')[[1]])}
hist(length_of_plays_word_C, freq=F, main="Distribution of # of words in Shakespeare's Comedies")
summary(length_of_plays_word_C)

length_of_plays_word_T <- vector(mode="numeric", length=10)
for(j in 1:10){length_of_plays_word_T[j]=length(
               strsplit(gsub(' {2,}',' ',clean_T[[j]][1]),' ')[[1]])}
hist(length_of_plays_word_T, freq=F, main="Distribution of # of words in Shakespeare's Tragedies")
summary(length_of_plays_word_T)

length_of_plays_word_H <- vector(mode="numeric", length=10)
for(j in 1:10){length_of_plays_word_H[j]=length(
               strsplit(gsub(' {2,}',' ',clean_H[[j]][1]),' ')[[1]])}
hist(length_of_plays_word_H, freq=F, main="Distribution of # of words in Shakespeare's Histories")
summary(length_of_plays_word_H)
```

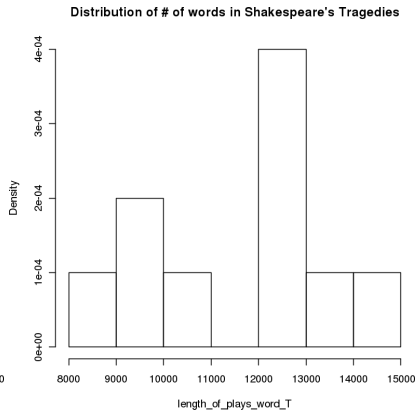


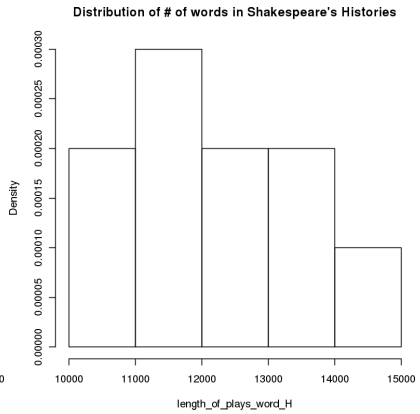





```
# summary(length_of_plays_char_C)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  47368   61656   70294   69295   74133   91107

# summary(length_of_plays_char_T)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  62703   67876   82889   79677   87843  100786

# summary(length_of_plays_char_H)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  71811   78630   83089   84155   89740   99953
```








```
# summary(length_of_plays_word_C)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   6930    9056   10198   10090   10849   13260

# summary(length_of_plays_word_T)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   9301   10246   12360   11759   12905   14718

# summary(length_of_plays_word_H)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10488   11535   12142   12270   13089   14475
```

Term-Document Matrices

We convert the corpora to TDM and remove terms that are **sparse** (too infrequent).

```
# Create TDMs
C_tdm <- tm::TermDocumentMatrix(clean_C)
T_tdm <- tm::TermDocumentMatrix(clean_T)
H_tdm <- tm::TermDocumentMatrix(clean_H)

# Remove sparse terms, with sparsity factor 75%
C_tdm <- tm::removeSparseTerms(C_tdm, 0.75)
T_tdm <- tm::removeSparseTerms(T_tdm, 0.75)
H_tdm <- tm::removeSparseTerms(H_tdm, 0.75)

# Print meta data
C_tdm
T_tdm
H_tdm

# Convert to matrices
C_m <- as.matrix(C_tdm)
T_m <- as.matrix(T_tdm)
H_m <- as.matrix(H_tdm)
```

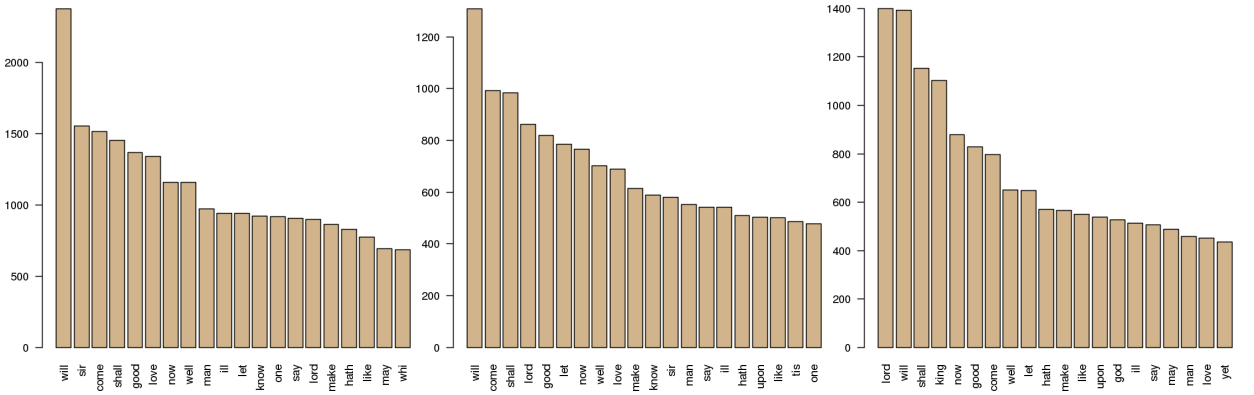
```
<<TermDocumentMatrix (terms: 2945, documents: 17)>>
Non-/sparse entries: 29556/20509
Sparsity           : 41%
Maximal term length: 12
Weighting          : term frequency (tf)

<<TermDocumentMatrix (terms: 3491, documents: 10)>>
Non-/sparse entries: 20699/14211
Sparsity           : 41%
Maximal term length: 12
Weighting          : term frequency (tf)

<<TermDocumentMatrix (terms: 3757, documents: 10)>>
Non-/sparse entries: 22335/15235
Sparsity           : 41%
Maximal term length: 14
Weighting          : term frequency (tf)
```

Barcharts

Next, we produce barcharts of the 20 most-frequent (sparsity-removed) terms in each corpus.

```
term_frequency_C <- rowSums(C_m)
term_frequency_T <- rowSums(T_m)
term_frequency_H <- rowSums(H_m)
```

```
# Sort term_frequency in descending order
term_frequency_C <- sort(term_frequency_C, decreasing=TRUE)
term_frequency_T <- sort(term_frequency_T, decreasing=TRUE)
term_frequency_H <- sort(term_frequency_H, decreasing=TRUE)

# Plot a barchart of the 20 most common words
barplot(term_frequency_C[1:20], col = "tan", las = 2)
barplot(term_frequency_T[1:20], col = "tan", las = 2)
barplot(term_frequency_H[1:20], col = "tan", las = 2)
```



comedies tragedies histories

Word Clouds, Commonality Clouds, and Comparison Clouds

It isn't always easy to read off the terms (we could also list them, of course) or to get a sense for how the corpora differ from one another with barcharts; word clouds (where the size of the word is linked to its frequency in the text) can help.

```
# Create word_freqs
word_freqs_C = data.frame(term_frequency_C)
word_freqs_C$term = rownames(word_freqs_C)
word_freqs_C = word_freqs_C[,c(2,1)]
colnames(word_freqs_C)=c("term","num")

word_freqs_T = data.frame(term_frequency_T)
word_freqs_T$term = rownames(word_freqs_T)
word_freqs_T = word_freqs_T[,c(2,1)]
colnames(word_freqs_T)=c("term","num")

word_freqs_H = data.frame(term_frequency_H)
word_freqs_H$term = rownames(word_freqs_H)
word_freqs_H = word_freqs_H[,c(2,1)]
colnames(word_freqs_H)=c("term","num")

# Create wordclouds
wordcloud::wordcloud(word_freqs_C$term, word_freqs_C$num, max.words=100, colors="red")
wordcloud::wordcloud(word_freqs_T$term, word_freqs_T$num, max.words=100, colors="blue")
wordcloud::wordcloud(word_freqs_H$term, word_freqs_H$num, max.words=100, colors="black")
```

comedies tragedies histories

To create **commonality clouds** and **comparison clouds**, we first create a list of all (cleaned) words in the comedies, tragedies, and histories, from the corpora `clean_C`, `clean_T`, and `clean_H`.

```
all_c = paste(clean_C[[1]][1],clean_C[[2]][1],clean_C[[3]][1],clean_C[[4]][1],
              clean_C[[5]][1],clean_C[[6]][1],clean_C[[7]][1],clean_C[[8]][1],
              clean_C[[9]][1],clean_C[[10]][1],clean_C[[11]][1],clean_C[[12]][1],
              clean_C[[13]][1],clean_C[[14]][1],clean_C[[15]][1],clean_C[[16]][1],
              clean_C[[17]][1],collapse=" ")
all_t = paste(clean_T[[1]][1],clean_T[[2]][1],clean_T[[3]][1],clean_T[[4]][1],
              clean_T[[5]][1],clean_T[[6]][1],clean_T[[7]][1],clean_T[[8]][1],
              clean_T[[9]][1],clean_T[[10]][1],collapse=" ")
all_h = paste(clean_H[[1]][1],clean_H[[2]][1],clean_H[[3]][1],clean_H[[4]][1],
              clean_H[[5]][1],clean_H[[6]][1],clean_H[[7]][1],clean_H[[8]][1],
              clean_H[[9]][1],clean_H[[10]][1],collapse=" ")
```

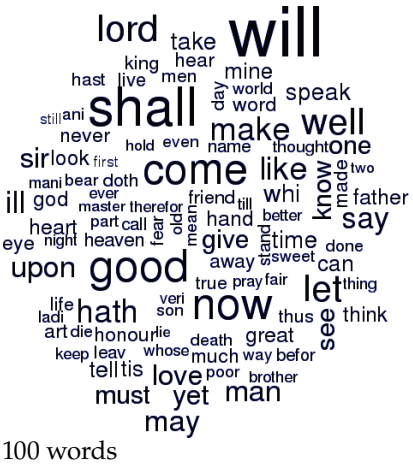
We join the terms as strings and put them into a single corpus.

```
ws_corpus = tm::VCorpus(tm::VectorSource(c(all_c,all_t,all_h))
tm::inspect(ws_corpus)
```

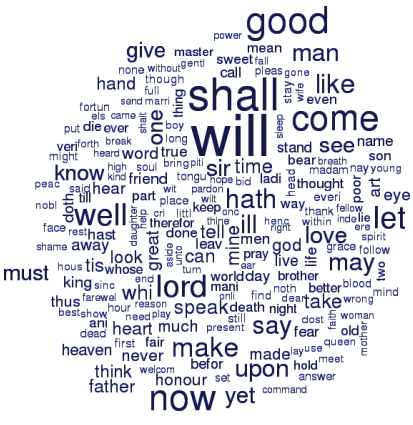
```
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 3

[[1]]
<<PlainTextDocument>>
Metadata:  7
Content:  chars: 1178027

[[2]]
<<PlainTextDocument>>
Metadata:  7
Content:  chars: 796779

[[3]]
<<PlainTextDocument>>
Metadata:  7
Content:  chars: 841554
```

Now we create a TDM for this corpus, which we cast as a matrix object before printing the commonality clouds (words shared across the three corpora), with 100, 200, and 500 words.

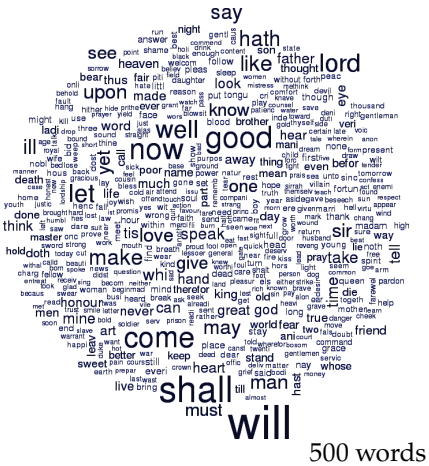
```
ws_tdm = tm::TermDocumentMatrix(ws_corpus)
colnames(ws_tdm) = c("Com.","Trag.","Hist.")
ws_m = as.matrix(ws_tdm)

wordcloud::commonality.cloud(ws_m, colors = "darkblue", max.words = 100)
wordcloud::commonality.cloud(ws_m, colors = "darkblue", max.words = 200)
wordcloud::commonality.cloud(ws_m, colors = "darkblue", max.words = 500)
```



100 words



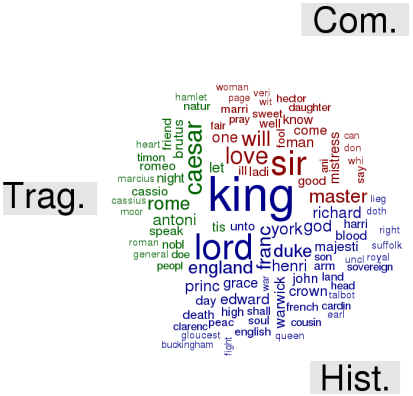
200 words



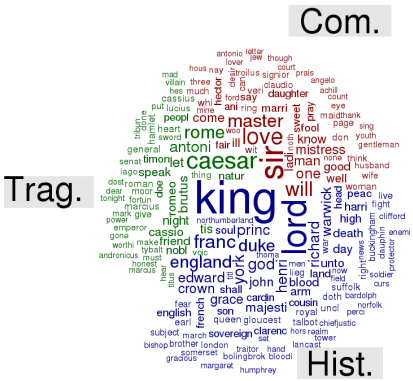
500 words

Comparison clouds can be produced using the same syntax.

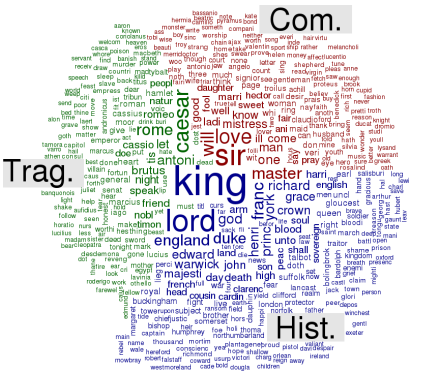
```
comp.colours = c("darkred","darkgreen","darkblue")
wordcloud::comparison.cloud(ws_m, colors = comp.colours, max.words = 100)
wordcloud::comparison.cloud(ws_m, colors = comp.colours, max.words = 200)
wordcloud::comparison.cloud(ws_m, colors = comp.colours, max.words = 500)
```



100 words



200 words



500 words

Pyramid Plots

We can also produce **pyramid plots** by first finding the terms that are common to any two corpora.

```
common_words_CT = subset(ws_m, ws_m[,1] > 0 & ws_m[,2] > 0)
dim(common_words_CT)
head(common_words_CT)
common_words_CH = subset(ws_m, ws_m[,1] > 0 & ws_m[,3] > 0)
dim(common_words_CH)
head(common_words_CH)
common_words_TH = subset(ws_m, ws_m[,2] > 0 & ws_m[,3] > 0)
dim(common_words_TH)
head(common_words_TH)
```

```
# common_words_CT
[1] 6438    3
         Docs
Terms     Com. Trag. Hist.
  abandon    2     3     0
  abat       5     5     4
  abate      3     2     3
  abbey      7     1     4
  abe        4     5     1
  abhor     14    16     3
```

```
# common_words_CH
[1] 6325    3
          Docs
Terms      Com. Trag. Hist.
  abandond    5     0     1
  abat        5     5     4
  abate       3     2     3
  abbey       7     1     4
  abe         4     5     1
  abet        1     0     1
```

```
# common_words_TH
[1] 5702    3
         Docs
Terms     Com. Trag. Hist.
  abat       5     5     4
  abate      3     2     3
  abbey      7     1     4
  abe        4     5     1
  abhor     14    16     3
  abhorrd    5     4     1
```

The differences in the number of times each token is used in each corpora can be computed as follows.

```
difference_CT = abs(common_words_CT[,1] - common_words_CT[,2])
difference_CH = abs(common_words_CH[,1] - common_words_CH[,3])
difference_TH = abs(common_words_TH[,2] - common_words_TH[,3])
```

Next, we bind these new counts to the respective `common_word` corpora, and order them along the differences.

```
common_words_CT = cbind(common_words_CT,difference_CT)
common_words_CT = common_words_CT[order(common_words_CT[,4],decreasing=TRUE),]
common_words_CH = cbind(common_words_CH,difference_CH)
common_words_CH = common_words_CH[order(common_words_CH[,4],decreasing=TRUE),]
common_words_TH = cbind(common_words_TH,difference_TH)
common_words_TH = common_words_TH[order(common_words_TH[,4],decreasing=TRUE),]
```

If we want to plot the top $n = 30$ words whose usage was the most different in each pair of corpora, we proceed as follows.

```
n=30
top_df_CT = data.frame(x = common_words_CT[1:n,1], y = common_words_CT[1:n,2],
                       labels = rownames(common_words_CT[1:n,]))
top_df_CH = data.frame(x = common_words_CH[1:n,1], y = common_words_CH[1:n,3],
                       labels=rownames(common_words_CH[1:n,]))
top_df_TH = data.frame(x = common_words_TH[1:n,2], y = common_words_TH[1:n,3], l
                       abels=rownames(common_words_TH[1:n,]))

top_df_CT; top_df_CH; top_df_TH
```

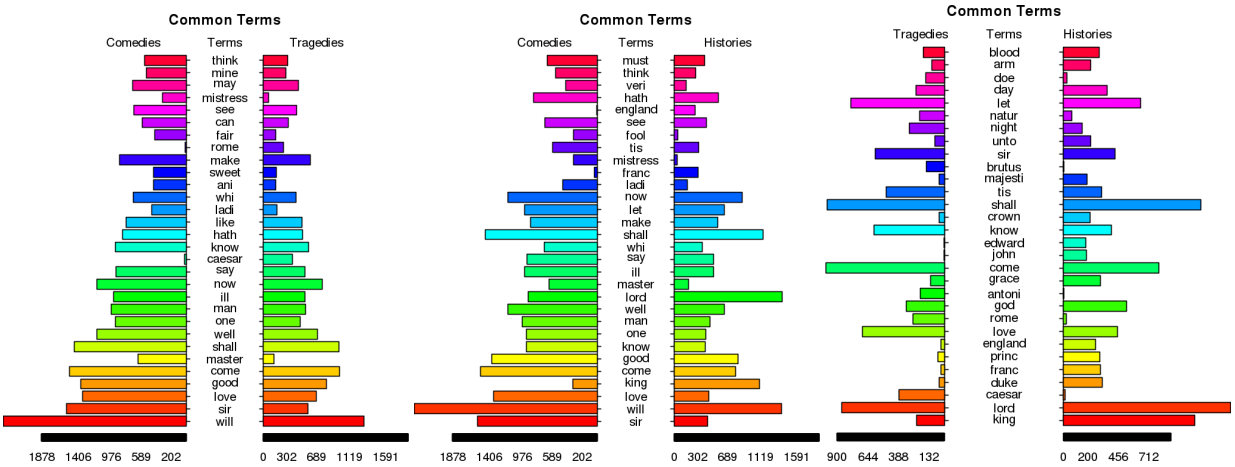
```
# top_df_CT
   x    y   labels
2378 1300     will
1554  581      sir
1343  688     love
1369  818     good
1518  984     come
1454  964    shall
 622  142   master
1159  700     well
 921  476      one
 973  552      man
 940  534      ill
1161  759      now
 909  540      say
  26  380   caesar
 922  587     know
 830  499     hath
 777  496     like
 688  413      whi
 447  180     ladi
 424  164      ani
 865  608     make
 422  170    sweet
 407  158     fair
  13  262     rome
 569  321      can
 676  429      see
 696  452      may
 309   67 mistress
 519  294     mine
 537  316    think
```

```
# top_df_CH
   x    y   labels
1554  434      sir
2378 1394     will
1343  451     love
 317 1103     king
1518  798     come
1369  829     good
 922  404     know
 921  407      one
 973  460      man
1159  650     well
 900 1400     lord
 622  186   master
 940  513      ill
 909  507      say
 688  359      whi
1454 1154    shall
 865  567     make
 940  649      let
1161  880      now
 447  171     ladi
  36  310    franc
 309   37 mistress
 583  319      tis
 306   44     fool
 676  414      see
   9  269  england
 830  570     hath
 412  152     veri
 537  279    think
 652  395     must
```

```
# top_df_TH
   x    y  labels
 231 1103    king
 861 1400    lord
 380   14  caesar
  42  327    duke
  27  310   franc
  54  305   princ
  31  269 england
 688  451    love
 262   25    rome
 318  527     god
 200    1  antoni
 112  309   grace
   4  194    john
 964 1154   shall
 984  798    come
   1  187  edward
 587  404    know
  43  223   crown
 483  319     tis
  43  196 majesti
 152    2  brutus
  77  226    unto
 581  434     sir
 293  154   night
 206   70   natur
 236  368     day
 157   28     doe
 777  649     let
 104  230     arm
 176  298   blood
```

Finally, we produce the pyramid plots themselves for the common terms that had the largest difference in usage for each pair of copora.

```
plotrix::pyramid.plot(top_df_CT$x,top_df_CT$y,labels=top_df_CT$labels,
             gap=500,top.labels=c("Comedies", "Terms", "Tragedies"), main="Common Terms",
             laxlab=NULL, raxlab=NULL, unit=NULL)

plotrix::pyramid.plot(top_df_CH$x,top_df_CH$y,labels=top_df_CH$labels,
             gap=500,top.labels=c("Comedies", "Terms", "Histories"), main="Common Terms",
             laxlab=NULL, raxlab=NULL, unit=NULL)

plotrix::pyramid.plot(top_df_TH$x,top_df_TH$y,labels=top_df_TH$labels,
             gap=500,top.labels=c("Tragedies", "Terms", "Histories"), main="Common Terms",
             laxlab=NULL, raxlab=NULL, unit=NULL)
```



27.4.4 Ham or Spam

In this example, we are going to use a classical SMS dataset where texts have been **classified** as ham/spam in order to build a model which can predict whether an incoming SMS is spam or ham based on its content.

Initializing the Environment

We will use the following R libraries:

- `tm` for text mining functions;
- `qdap` for some text processing functions;
- `e1071` for the naive Bayes and support vector machines methods;
- `dplyr` for tidyverse processing;
- `tidytext` for tidyverse analysis;
- `ggplot2` for tidyverse plotting, and
- `psych` for regular plotting.

Importing and Exploring the Data

We load the dataset as usual.

```
ham.spam <- read.csv("SMSSpamCollection.csv", sep=",")
ham.spam$Msg <- as.character(ham.spam$Msg)
```

The dataset consists of 5574 observations and 3 variables/features: SMS messages, length of the messages, and whether they are ham (+) or spam (−).

```
str(ham.spam)
```

```
'data.frame':  5574 obs. of  3 variables:
 $ SpamOrHam: Factor w/ 2 levels "ham","spam": 1 1 2 1 1 2 1 1 2 2 ...
 $ Msg      : chr  "Go until jurong point, crazy.. Available only in bugis n great world la ..." ...
 $ length   : int  111 29 155 49 61 148 77 160 158 154 ...
```

We print a few SMS (one spam and one ham) to get a better idea as to the contents of the text.

```
ham.spam$Msg[12]
ham.spam$Msg[4444]
```

```
[1] "SIX chances to win CASH! From 100 to 20,000 pounds txt>
     CSH11 and send to 87575. Cost 150p/day, 6days, 16+
     TsandCs apply Reply HL 4 info"
[1] "Dear i am not denying your words please"
```

No human would send the first SMS to another person – we all recognize it as spam; the second one seems more legitimate – we have all had conversations of this nature with our partners. The SMS labels confirms the suspicion.

```
ham.spam$SpamOrHam[12]
ham.spam$SpamOrHam[4444]
```

```
[1] "spam"
[1] "ham"
```

What is the distribution of ham/spam messages in the dataset?

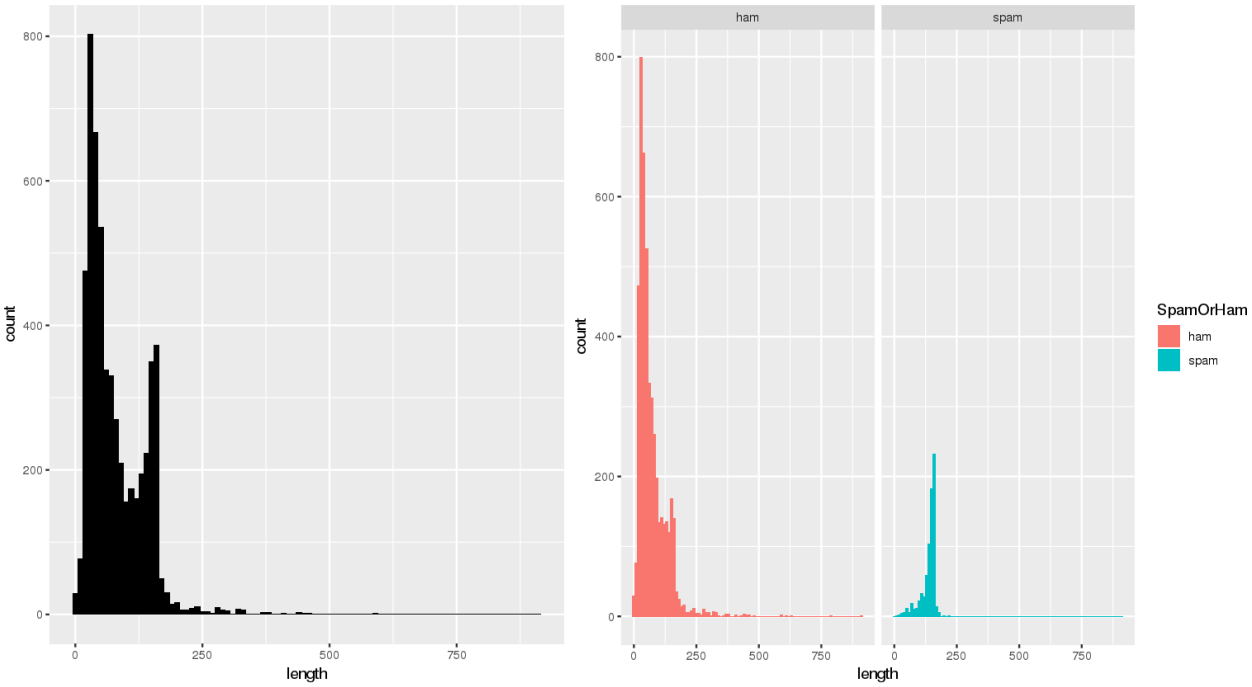
```
table(ham.spam$SpamOrHam)
prop.table(table(ham.spam$SpamOrHam))
```

```
 ham spam
4827  747

      ham      spam
0.8659849 0.1340151
```

Before we dive right into text analysis, can we say anything about the SMS categories just by looking at the lengths of the SMS? Are the examples above representative of spam and ham messages?

Should we expect spam messages to be longer than ham messages, in general? The distribution of message lengths is **bimodal** – does that mean anything?

```
library(ggplot2)
ggplot(ham.spam, aes(length)) +
       geom_histogram(binwidth=10)
ggplot(ham.spam, aes(length, fill = SpamOrHam)) +
       geom_histogram(binwidth=10) +
       facet_wrap(~SpamOrHam)
```



There are more ham messages than spam messages in the dataset, obviously but the height of the distributions is not as important as the shape: in the absence of more information, if the SMS has 150 characters or more, say, it would not be the most unreasonable thing in the world to suspect that it could be spam... but the contents of the message also have to count for something, right?

Creating the Corpus

In order to do text classification, we first need to prepare a document-term matrix. The `tm` functionality from the previous examples does the trick.

```
(SMS.corpus <- tm::VCorpus(tm::VectorSource(ham.spam$Msg)))
```

```
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 5574
```

We take a quick peek at the first 4 entries to make sure that everything is as it should be.

```
sapply(SMS.corpus[1:4], function(x){x$content})
```

```
[1] Go until jurong point, crazy.. Available only in bugis n great world la e buffet...
    Cine there got amore wat...
[2] Ok lar... Joking wif u oni...
[3] Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121
    to receive entry question(std txt rate)T&C's apply 08452810075over18's
[4] U dun say so early hor... U c already then say...
```

Cleaning the Data

As is **almost always** the case, the next step is to clean the corpus. Note that the order in which the cleaning steps are performed may affect the final form of the cleaned corpus.

```
clean_corpus <- function(corpus){
  corpus <- tm::tm_map(corpus, tm::content_transformer(qdap::replace_abbreviation))
  corpus <- tm::tm_map(corpus, tm::removePunctuation)
  corpus <- tm::tm_map(corpus, tm::removeNumbers)
  corpus <- tm::tm_map(corpus, tm::stemDocument)
  corpus <- tm::tm_map(corpus, tm::content_transformer(tolower))
  corpus <- tm::tm_map(corpus, tm::stripWhitespace)
  corpus <- tm::tm_map(corpus, tm::removeWords, c(tm::stopwords("en")))
  return(corpus)
}

SMS.corpus.clean <- clean_corpus(SMS.corpus)
sapply(SMS.corpus.clean[1:4], function(x){x$content})
```

```
[1] go  jurong point crazi availabl onli  bugi n great world la e buffet cine  got amor wat
[2] ok lar joke wif u oni
[3] free entri   wkli comp  win fa cup final tkts st may text fa   receiv entri questionstd
    txt ratetc appli
[4] u dun say  earli hor u c alreadi  say
```

In the case of spam detection, there could be reasons why we might not want to be too drastic at the cleaning stage: removing the numerals in `08452810075over18's` in the 3rd SMS, for instance, removes an important indicator of spam.

Creating the DTM

We want to classify **documents**, so we need a DTM (and not a TDM) representation of the text dataset on which to apply classifiers.

```
SMS.DTM <- tm::DocumentTermMatrix(SMS.corpus.clean)
tm::inspect(SMS.DTM[1:15,20:30])
```

```
<<DocumentTermMatrix (documents: 20, terms: 11)>>
Non-/sparse entries: 17/203
Sparsity           : 92%
Maximal term length: 11
Weighting          : term frequency (tf)

Sample             :
    Terms
Docs aaniy aaooooright aathilov aathiwher abbey abdomen abeg abelu aberdeen abi abil
  1      0           0        0         0     0       0    0     0        0   0    0
  2      0           0        0         0     0       0    0     0        0   0    0
  3      0           0        0         0     0       0    0     0        0   0    0
  4      0           0        0         0     0       0    0     0        0   0    0
```

5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0

Text Visualization

Before we start with classification proper, let us visualize the frequent terms of both the spam and the ham classes.

```
spam.cloud <- which(ham.spam$SpamOrHam == "spam")
ham.cloud <- which(ham.spam$SpamOrHam == "ham")
wordcloud::wordcloud(SMS.corpus.clean[ham.cloud],
                     min.freq=50,colors="steelblue")
wordcloud::wordcloud(SMS.corpus.clean[spam.cloud],
                     min.freq=50,colors="darkred")
```

ham SMS

spam SMS

We can also identify document-specific high-information terms using the **tf-idf** weighting.

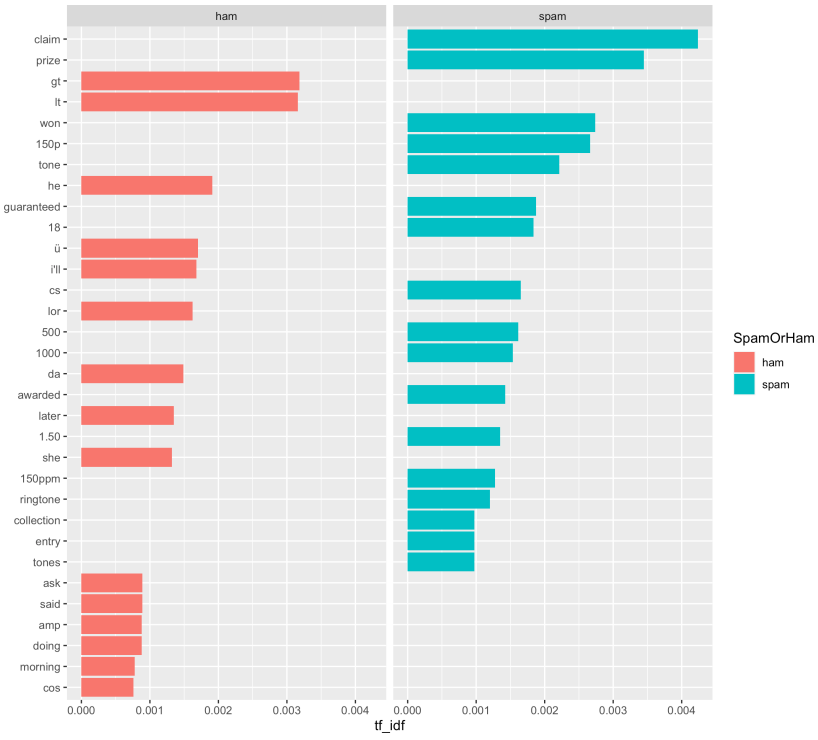
```
SMS_words <- ham.spam |>
  tidytext::unnest_tokens(word,Msg) |>
  dplyr::count(SpamOrHam,word, sort=TRUE) |>
  dplyr::ungroup()
```

```
SMS_distinctive <- SMS_words |>
  tidytext::bind_tf_idf(word,SpamOrHam,n) |>
  dplyr::arrange(desc(tf_idf)) |>
  dplyr::mutate(word = factor(word,
                levels = rev(unique(word)))) |>
  dplyr::group_by(SpamOrHam) |>
  dplyr::top_n(15,tf_idf) |> dplyr::ungroup()

head(SMS_distinctive)
```

```
# A tibble: 6 × 6
  SpamOrHam word      n      tf   idf  tf_idf
  <chr>     <fct> <int>   <dbl> <dbl>   <dbl>
1 spam      claim   113 0.00611 0.693 0.00424
2 spam      prize    92 0.00498 0.693 0.00345
3 ham       gt      318 0.00459 0.693 0.00318
4 ham       lt      316 0.00456 0.693 0.00316
5 spam      won      73 0.00395 0.693 0.00274
6 spam      150p     71 0.00384 0.693 0.00266
```

A simple visual identifies terms that are specific to spam/ham SMS.

```
library(ggplot2)
ggplot(SMS_distinctive, aes(word,tf_idf, fill = SpamOrHam)) +
  geom_col(show.legend=TRUE) + labs(x=NULL, y="tf_idf") +
  facet_wrap(~SpamOrHam) + coord_flip()
```



Training/Testing Data

Classifiers need to be trained on a subset of the data and tested/evaluated on the complement to avoid overfitting the model. There's no steadfast rule, but a 70%/30% split is often applied.[34]

34: When attempting to replicate what follows, remember that we did not set a seed, and so that your results could be somewhat different.

```
ind = sample(1:nrow(ham.spam), size=0.7*nrow(ham.spam))
spam.train = subset(ham.spam[ind,], SpamOrHam == "spam")
ham.train = subset(ham.spam[ind,], SpamOrHam == "ham")
ham.spam.train.labels <- ham.spam[ind,]$SpamOrHam
ham.spam.test.labels <- ham.spam[-ind,]$SpamOrHam
```

We can verify that the training/testing sets are representative of the full dataset (in terms of the target labels, at least).

```
prop.table(table(ham.spam.train.labels))
prop.table(table(ham.spam.test.labels))
prop.table(table(ham.spam$SpamOrHam))
```

```
ham.spam.train.labels
      ham      spam
0.8664445 0.1335555

ham.spam.test.labels
      ham      spam
0.8649133 0.1350867

      ham      spam
0.8659849 0.1340151
```

Next, we convert the training/testing messages to DTM and corpora.

```
SMS.DTM.train <- SMS.DTM[ind,]
SMS.DTM.test <- SMS.DTM[-ind,]
SMS.corpus.clean.train <- SMS.corpus.clean[ind]
SMS.corpus.clean.test <- SMS.corpus.clean[-ind]
```

We also need to select the **features** (in this case, the terms) to include in the model – otherwise, there would be too much information to consider and the **curse of dimensionality** rears is ugly head (see Chapter 23). We can use the list of all terms that appear at least 10 times, say, in the training messages, for instance.[35]

35: Another approach could be to remove **sparse terms**.

```
Freq.Terms <- tm::findFreqTerms(SMS.DTM.train,10)
SMS.DTM.Freq.Terms.train <- SMS.DTM.train[,Freq.Terms]
SMS.DTM.Freq.Terms.test <- SMS.DTM.test[,Freq.Terms]
length(Freq.Terms)
Freq.Terms[1:100]
```

```
[1] 614
```

```
[1] 'crazi' 'got' 'great' 'onli' 'point' 'wat' 'world' 'joke' 'lar' 'wif' 'appli' 'entri'
[13] 'final' 'free' 'may' 'receiv' 'text' 'txt' 'win' 'wkli' 'alreadi' 'dun' 'earli' 'say'
[25] 'around' 'dont' 'goe' 'live' 'think' 'though' 'back' 'freemsg' 'fun' 'hey' 'like'
[36] 'now' 'send' 'still' 'week' 'word' 'xxx' 'brother' 'even' 'speak' 'treat' 'caller'
[47] 'copi' 'friend' ...
```

We also need to **categorize** the features so that the data can eventually be fed into a naïve Bayes classifier, say; the absence of a frequent term in a SMS message is denoted by "No", while it's presence is denoted by "Yes".

```
yes.no <- function(x){
    y <- ifelse(x>0,1,0)
    y <- factor(y,levels=c(0,1),labels=c("No","Yes"))
    return(y)
}
```

The (reduced) training/testing sets thus look like:

```
SMS.train <- apply(SMS.DTM.Freq.Terms.train,2,yes.no)
SMS.test <- apply(SMS.DTM.Freq.Terms.test,2,yes.no)
head(SMS.train)
head(SMS.test)
```

```
# SMS.train
     crazi  got great onli point    wat ... auction lei
3151    No    No    No   No     No    No ...      No   No
1572    No   Yes    No   No     No    No ...      No   No
3200    No    No    No   No     No   Yes ...      No   No
1805    No    No    No   No     No    No ...      No   No
1201    No    No    No   No     No    No ...      No   No
534     No   No    No   No     No    No ...      No  No

# SMS.text
     crazi  got great onli point    wat ... auction lei
3       No   No    No   No     No    No ...      No   No
7       No   No    No   No     No    No ...      No   No
9       No   No   Yes   No     No    No ...      No   No
11      No   No    No   No     No    No ...      No   No
13      No   No    No   No     No    No ...      No   No
14      No   No    No   No     No    No ...      No  No
```

Naive Bayes Classifier

We now apply the `naiveBayes()` function from the `R` library `e1071`.[36] Notice the syntax: we apply `naiveBayes` to the training data `SMS.train` and the target variable is the ham/spam label in the training subset (`ham.spam.train.labels`). The `laplace=1` option instructs `naiveBayes()` to look "a little bit harder" into the data, while the `CV=10` option selects 10 cross-validations replicates.

36: See Section 21.4.4 for details.

```
SMS.classifier.NB <- e1071::naiveBayes(SMS.train,ham.spam.train.labels,laplace=1,CV=10)
summary(SMS.classifier.NB)
attributes(SMS.classifier.NB)
```

```
       Length Class  Mode
apriori     2  table  numeric
tables    614  -none- list
levels      2  -none- character
isnumeric 614  -none- logical
call        5  -none- call

$names
    'apriori' 'tables' 'levels' 'isnumeric' 'call'
$class
    'naiveBayes'
```

We can now feed the testing data (`SMS.test`) into the model (`SMS.classifier.NB`) with the help of the `predict()` function.

```
SMS.test.pred.NB <- predict(SMS.classifier.NB,
                            newdata = SMS.test)
table(SMS.test.pred.NB,ham.spam.test.labels)
prop.table(table(SMS.test.pred.NB,ham.spam.test.labels))
```

```
                ham.spam.test.labels
SMS.test.pred.NB  ham spam
            ham  1444   26
            spam    3  200

                ham.spam.test.labels
SMS.test.pred.NB         ham        spam
            ham  0.863120143 0.015540944
            spam 0.001793186 0.119545726
```

These **confusion matrices** are not bad at all! Is there a difference in the representations of mislabeled and sucessfully labeled test SMS?

```
Missed=SMS.test[which(SMS.test.pred.NB!=ham.spam.test.labels),]
Succesful=SMS.test[which(SMS.test.pred.NB==ham.spam.test.labels),]
table(Missed)
table(Succesful)
```

```
Missed
   No   Yes
17653   153

Succesful
     No     Yes
1000706    8710
```

The ratios are basically the same in both instances. We can also take a look at the original text of a few mislabeled and successfully labeled test SMS.

```
head(ham.spam[as.numeric(rownames(Missed)),]$Msg)
head(ham.spam[as.numeric(rownames(Succesful)),]$Msg)
```

```
# some mispredicted SMS
[1] 'England v Macedonia - dont miss the goals/team news. Txt ur national team to 87077 eg
    ENGLAND to 87077 Try:WALES, SCOTLAND 4txt/ú1.20 POBOXox36504W45WQ 16+'
[2] 'U 447801259231 have a secret admirer who is looking 2 make contact with U-find out who
    they R*reveal who thinks UR so special-call on 09058094597'
[3] 'SMS. ac Blind Date 4U!: Rodds1 is 21/m from Aberdeen, United Kingdom. Check Him out
    http://img. sms. ac/W/icmb3cktz8r7!-4 no Blind Dates send HIDE'
[4] 'XCLUSIVE@CLUBSAISAI 2MOROW 28/5 SOIREE SPECIALE ZOUK WITH NICHOLS FROM PARIS.FREE ROSES
    2 ALL LADIES !!! info: 07946746291/07880867867 '
[5] 'Its a valentine game. . . Send dis msg to all ur friends. .. If 5 answers r d same then
    someone really loves u. Ques- which colour suits me the best?rply me'
[6] 'Hi I\'m sue. I am 20 years old and work as a lapdancer. I love sex. Text me live - I\'m
    i my bedroom now. text SUE to 89555. By TextOperator G2 1DA 150ppmsg 18+'

# some correctly predicted SMS
[1] 'Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to
    receive entry question(std txt rate)T&C\'s apply 08452810075over18\'s'
[2] 'Even my brother is not like to speak with me. They treat me like aids patent.'
[3] 'WINNER!! As a valued network customer you have been selected to receivea £900 prize reward!
    To claim call 09061701461. Claim code KL341. Valid 12 hours only.'
[4] 'I\'m gonna be home soon and i don\'t want to talk about this stuff anymore tonight, k? I\'ve
    cried enough today.'
[5] 'URGENT! You have won a 1 week FREE membership in our £100,000 Prize Jackpot! Txt the word:
    CLAIM to No: 81010 T&C www.dbuk.net LCCLTD POBOX 4403LDNW1A7RW18'
[6] 'I\'ve been searching for the right words to thank you for this breather. I promise i wont
    take your help for granted and will fulfil my promise. You have been wonderful and a blessing
    at all times.'
```

Support Vector Machines

We can use other classifiers – `e1071` also implements support vector machines.[37] The principle is the same, although we do not need to first categorize the features (as we had to do for naïve Bayes.

37: See Section for 21.4.2 for details.

The `cost` parameter is related to the price of allowing **misclassifications**: higher values attempt to minimize such misclassifications, but the resulting model tends to lead to overfitting (a little flexibility in re: misclassfications is a good thing in the long run).

There is some evidence to suggest that the choice of the `"linear"` kernel is preferable when dealing with text data (as opposed to gaussian for some numerical data), but other kernels can also be used.

The steps proceeds as in naïve Bayes (although we demonstrate the `classAgreement` functionality for evaluation the classifier). You can get details on the method by querying the help files.

```
SMS.classifier.SVM <- e1071::svm(as.factor(ham.spam.train.labels) ~.,
                                 data=as.data.frame(as.matrix(SMS.DTM.Freq.Terms.train)),
                                 type="C-classification", cost=10, kernel="linear")
```

```
summary(SMS.classifier.SVM)
attributes(SMS.classifier.SVM)
```

```
Call:
svm(formula = ham.spam.train.labels ~ ., data = as.data.frame(as.matrix(SMS.DTM.Freq.Terms.train)),
    type = "C-classification", cost = 10, kernel = "linear")

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10
      gamma:  0.001628664

Number of Support Vectors:  424

 ( 299 125 )

Number of Classes:  2


Levels:
 ham spam

$names
    'call' 'type' 'kernel' 'cost' 'degree' 'gamma' 'coef0' 'nu' 'epsilon' 'sparse' 'scaled'
    'x.scale' 'y.scale' 'nclasses' 'levels' 'tot.nSV' 'nSV' 'labels' 'SV' 'index' 'rho' 'compprob'
    'probA' 'probB' 'sigma' 'coefs' 'na.action' 'fitted' 'decision.values' 'terms'

$class
    'svm.formula' 'svm'
```

We can now feed the testing data into the model with the help of the `predict()` function.

```
SMS.test.pred.SVM <- predict(SMS.classifier.SVM,as.data.frame(as.matrix(SMS.DTM.Freq.Terms.test)))
summary(SMS.test.pred.SVM)
```

```
 ham spam
1430  243
```

Of course, it may not be sufficient to know how many SMS are predicted to be ham and/or spam – we might want to know if individual SMS are correctly predicted.

```
(confusion.matrix = table(pred = SMS.test.pred.SVM, true = ham.spam.test.labels))
e1071::classAgreement(confusion.matrix,match.names=TRUE)
```

```
      true
pred    ham spam
  ham  1401   29
  spam   46  197
```

```
$diag
[1] 0.95517

$kappa
[1] 0.81406

$rand
[1] 0.91431

$crand
[1] 0.76592
```

More information on this last function can be obtained by typing in `?e1071::classAgreement` at the prompt.

SVMs with PCA

Finally, we re-visit the SVM model by first reducing the DTM data to its first 4 **principal components** in order to try to mitigate the effects of the curse of dimensionality and to introduce a more complete classification workflow:[38]

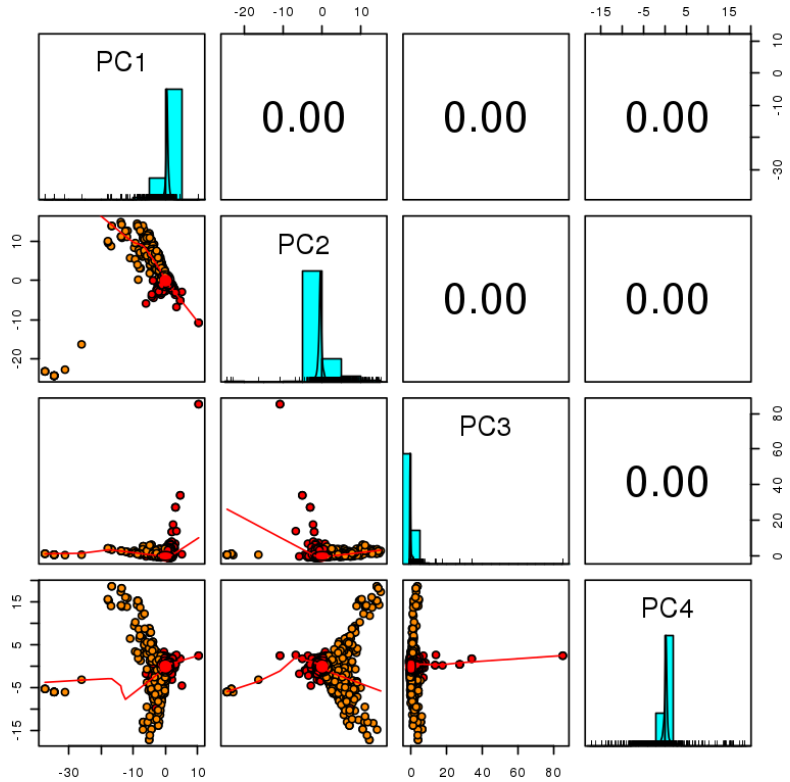
38: See Chapter 23 for more details.

1. compute the principal composition of the training data;
2. set-up the formulas for going back and forth between the original data and the rotated (reduced) data;
3. plot the classes against the first 4 principal components (arbitrary);
4. express the test data in the training PCA universe;
5. tune the SVM model (run some preliminary code to determine the optimal choice of model parameters);
6. train the SVM model;
7. fit the SVM model to testing data, and
8. evaluate the model.

```
# 1: find the PCs of the training data and only keep the first 4, say
SMS.train.pca <- prcomp(as.data.frame(as.matrix(SMS.DTM.Freq.Terms.train)),
       center = TRUE, scale = TRUE)
SMS.train.pca.reduced = SMS.train.pca$x
SMS.train.pca.reduced[,5:ncol(SMS.train.pca.reduced)] = 0   # setting PCs 5-end to 0
```

```
# 2: going back and forth between the original data and the rotated data
SMS.train.recover.full = exp(t(t(SMS.train.pca$x %*% t(SMS.train.pca$rotation)) *
                         SMS.train.pca$scale + SMS.train.pca$center))
SMS.train.recover.pca = exp(t(t(SMS.train.pca.reduced %*% t(SMS.train.pca$rotation)) *
                         SMS.train.pca$scale + SMS.train.pca$center))

# put the data in a data frame
SMS.train.pca.reduced.model = SMS.train.pca.reduced
SMS.train.pca.reduced.model = data.frame(SMS.train.pca.reduced, ham.spam.train.labels)

training.pca = SMS.train.pca.reduced.model[,c(1,2,3,4)]
```

```
# 3: plot the classes in the training data set against the first 4 PCs
psych::pairs.panels(SMS.train.pca$x[,1:4], pch=21,
                    bg=rainbow(11)[unclass(ham.spam.train.labels)])
```




```
# 4: put test data in the PCA context
SMS.test.pca <- scale(as.data.frame(as.matrix(SMS.DTM.Freq.Terms.test)),
      SMS.train.pca$center, SMS.train.pca$scale) %*% SMS.train.pca$rotation

SMS.test.pca.reduced <- SMS.test.pca
SMS.test.pca.reduced[,5:ncol(SMS.train.pca.reduced)] = 0

SMS.test.pca.reduced.model = data.frame(SMS.test.pca.reduced)
```

```
# 5: tune SVM model to find a good choice for the cost parameter C
tuning.pca <- e1071::tune(e1071::svm, train.x=training.pca,
                          train.y=as.factor(ham.spam.train.labels), kernel="linear",
                          ranges=list(cost=10^(-3:3)))
print(tuning.pca)  # optimal parameters
```

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost
   10

- best performance: 0.0302492
```

```
# 6: train the SVM model
SMS.classifier.SVM.pca <- e1071::svm(as.factor(ham.spam.train.labels) ~., data=training.pca,
                                     type="C-classification", cost=10, kernel="linear")
summary(SMS.classifier.SVM.pca)
```

```
Call: svm(formula = ham.spam.train.labels ~ ., data = training.pca, type = "C-classification",
    cost = 10, kernel = "linear")


                                Parameters:
                                   SVM-Type:  C-classification
                                 SVM-Kernel:  linear
                                       cost:  10
                                      gamma:  0.25

                                Number of Support Vectors:  316

                                 ( 158 158 )

                                Number of Classes:  2

                                Levels:
                                 ham spam
```

```
# 7: fit SVM to test data
SMS.test.pred.SVM.pca <- predict(SMS.classifier.SVM.pca,SMS.test.pca.reduced)
summary(SMS.test.pred.SVM.pca)   # provide a summary of the predicted values
```

```
                                 ham spam
                                1430  243
```

```
# 8: evaluate the fitted model
(confusion.matrix.pca = table(pred = SMS.test.pred.SVM.pca, true = ham.spam.test.labels))
e1071::classAgreement(confusion.matrix.pca,match.names=TRUE)
```

```
                                     true
                                pred    ham spam
                                  ham  1438   40
                                  spam    9  186

                                $diag
                                [1] 0.97071

                                $kappa
                                [1] 0.86696

                                $rand
                                [1] 0.94310

                                $crand
                                [1] 0.83437
```

How does the model on the reduced data compare to the original SVM model?

27.4.5 NHL Game Recaps (Reprise)

In this section, we revisit the NHL game recaps of Section 27.4.1, this time to **cluster** the terms in the documents. Clustering is an unsupervised learning technique that can be used to determine which entities or objects, typically represented by rows in the data, are similar to each other. For text mining, we can use clustering to determine which documents are similar to one another (based on their term signature), for instance.[39]

For the purpose of this example, we will use **hierarchical clustering**, but any other clustering approach would be just as acceptable.[40]

39: We will discuss **topic modeling**, another unsupervised learning application to text data, in Chapter 32.

40: See Chapter 22 for details.

Review of Hierarchical Clustering in R

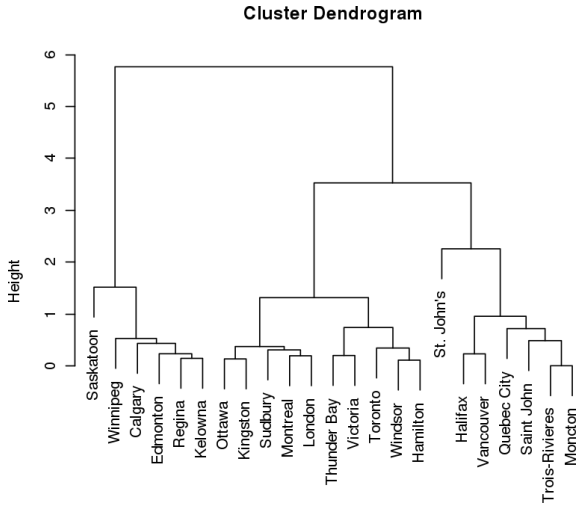
Let's start with a simple example of hierarchical clustering using numeric data. The three lines of data below represent days of rainfall in a number of Canadian cities. Can we use hierarchical clustering to get a sense of which cities are similar to each other?

```
city = c("Montreal","Ottawa","Toronto","Quebec City","Kingston","Trois-Rivieres","Windsor",
         "Hamilton","London","Halifax","Moncton","Saint John","St. John's","Sudbury",
         "Thunder Bay","Winnipeg","Saskatoon","Regina","Calgary","Edmonton","Kelowna",
         "Vancouver","Victoria")
rainfall = c(1000,920,831,1184,960,1123,935,897,1012,1468,1124,1295,1534,903,684,521,365,
             390,456,419,345,1457,705)
days = c(163,161,145,175,159,161,150,149,168,162,161,158,212,167,143,125,87,118,112,123,
         120,168,148)
```

The code below starts by creating and scaling a data frame for the data. Then `hclust()` builds the clustering information, which `plot()` then uses to display the **clustering dendrogram**.[41]

41: We use `hclust`'s Euclidean dissimilarity and complete linkage defaults.

```
rain.data <- data.frame(city, rainfall,days)
dist.rain <- dist(scale(rain.data[,2:3]))
hc <- hclust(dist.rain) # distances as hc object
plot(hc, labels = rain.data$city)
```



It is not surprising to see that Ottawa and Kingston are very similar when it comes to rainfall totals and number of rainy days, as are Hamilton, Windsor, and Toronto (each group of cities being in the same general region), but it might be surprising to see Halifax and Vancouver as being similar – until we remember that they are both coastal cities.

The Return of the Text-Rich Hockey Dataset

Now that we have seen a simple example of how hierarchical clustering works with numerical data, we turn our attention back to text data, and the hockey dataset of Section 27.4.1.

As we did then, we clean the data (removing stop words, etc.) from the Sens games, at which point we create a **term-document matrix** (TDM) from the cleaned text data.

With the TDM in hand, we can switch over to clustering the data, once we have generated a **distance matrix** (DM),[42] which we use to hierarchically cluster the data, and generate the text data's cluster dendrogram.

42: We could consider either document or term similarity for the DM; here we will use term similarity.

```
recaps <- read.csv(file="Recap_data_first_pass_utf8.csv", header=TRUE, sep=",",
                   stringsAsFactors=FALSE)

# Isolate the text recaps
AP.recaps <- recaps$AP.Recap

# Make a vector source
AP.recaps.source <- tm::VectorSource(AP.recaps)

# Make a volatile corpus
AP.recaps.corpus <- tm::VCorpus(AP.recaps.source)

# Create a customized function to clean the corpus
clean_corpus_Sens <- function(corpus){
  corpus <- tm::tm_map(corpus, content_transformer(qdap::replace_abbreviation))
  corpus <- tm::tm_map(corpus, tm::removePunctuation)
  corpus <- tm::tm_map(corpus, tm::removeNumbers)
  corpus <- tm::tm_map(corpus, tm::stemDocument)
  corpus <- tm::tm_map(corpus, tm::content_transformer(tolower))
  corpus <- tm::tm_map(corpus, tm::stripWhitespace)
  corpus <- tm::tm_map(corpus, tm::removeWords, c(tm::stopwords("en"), "game", "first",
                       "second", "third", "Ottawa", "Senators"))
  return(corpus)
}

# Apply the customized function to the corpus
clean_corp.AP.recaps <- clean_corpus_Sens(AP.recaps.corpus)

# Create a TDM from the corpus
AP.recaps_tdm <- tm::TermDocumentMatrix(clean_corp.AP.recaps)

# Remove sparse terms
AP.recaps_tdm_50 <- tm::removeSparseTerms(AP.recaps_tdm,sparse=.5)
```

```
# What effect did this have on the number of terms?
AP.recaps_tdm_50
```

```
<<TermDocumentMatrix (terms: 82, documents: 101)>>
Non-/sparse entries: 5806/2476
Sparsity           : 30%
Maximal term length: 8
Weighting          : term frequency (tf)
```

```
# Convert AP.recaps_tdm to a matrix: AP.recaps_m
AP.recaps_tdm_50_m <- as.matrix(AP.recaps_tdm_50)

# Save the term document matrix as a data frame
TDM_50.df <- as.data.frame(AP.recaps_tdm_50_m)
TDM_50.df
```

```
         1   2   3   4   5   6   7   8   9  10  ... 92  93  94  95  96  97  98  99 100 101
also     2   0   1   2   1   1   1   1   2   1  ...  4   1   1   1   0   2   0   1   2   0
anderson 3   3   0   2   3   5   1  13   6   1  ...  4   3   5   4   5   2   2   2  10   6
assist   1   0   1   2   0   2   2   0   2   0  ...  4   1   2   1   1   0   2   4   0   2
back     0   3   0   1   0   0   2   1   1   0  ...  4   5   1   1   3   2   1   1   4   1
beat     1   3   1   3   1   1   1   1   1   0  ...  2   1   2   0   0   3   1   3   2   0
befor    1   0   1   4   0   3   2   0   0   1  ...  1   0   0   1   0   2   0   0   0   1
boucher  2   2   0   0   3   2   0   0   2   1  ...  4   3   0   4   1   2   1   3   3   1
came     2   1   0   1   2   2   0   1   0   3  ...  2   2   1   1   2   2   0   1   1   1
chanc    0   2   0   0   0   0   2   1   1   2  ...  2   0   5   2   0   2   0   1   1   0
coach    4   3   0   1   2   1   1   0   3   5  ...  2   2   1   2   1   3   2   2   2   2
come     1   0   0   1   0   1   1   0   1   2  ...  2   1   1   1   0   0   0   0   1   1
craig    1   1   0   1   1   1   1   1   1   1  ...  1   1   1   1   1   1   1   1   1   2
didnt    1   2   0   1   2   2   1   2   0   0  ...  1   1   1   1   3   1   1   1   5   1
end      0   0   0   0   0   0   0   1   2   0  ...  1   1   0   0   4   1   0   0   0   1
erik     2   1   0   1   1   0   1   1   0   2  ...  1   1   1   0   0   1   1   1   1   2
final    0   0   0   1   0   0   0   2   0   1  ...  2   2   4   4   3   2   1   6   3   5
five     1   0   0   1   0   1   3   0   1   2  ...  0   1   0   1   1   0   1   1   0   0
four     2   0   1   1   0   1   0   0   2   1  ...  3   1   0   2   0   2   1   3   2   3
gave     0   1   1   2   0   0   2   1   0   0  ...  0   2   0   2   0   1   1   0   2   0
get      3   2   0   3   2   3   1   0   1   2  ...  2   3   2   5   3   3   3   2   4   2
give     1   1   0   4   1   0   0   1   0   0  ...  0   1   0   3   1   2   0   2   0   2
goal     9   2   9  17   4   4   6   1   3   4  ...  8  13   5   5   1  13   5   4   5   4
good     1   0   0   2   1   4   1   0   3   3  ...  3   1   2   2   1   0   5   3   1   0
got      1   2   0   3   5   0   1   1   0   1  ...  4   2   5   1   0   4   2   1   2   1

        ...  ...  ...  ...  ...  ...  ...  ...  ...  ... ...  ...  ...  ...  ...  ...  ...  ...  ...  ...

two      1   2   3   4   0   4   4   3   0   2  ...  9   5   2   2   2   1   6   3   4   3
way      1   3   0   0   1   2   1   1   2   1  ...  0   2   1   2   2   0   1   1   1   5
went     0   0   0   1   0   1   1   0   1   0  ...  1   1   1   1   2   1   3   1   0   0
win      1   4   0   2   0   0   2   2   1   0  ...  3   6   0   1   2   3   0   1   1   1
```

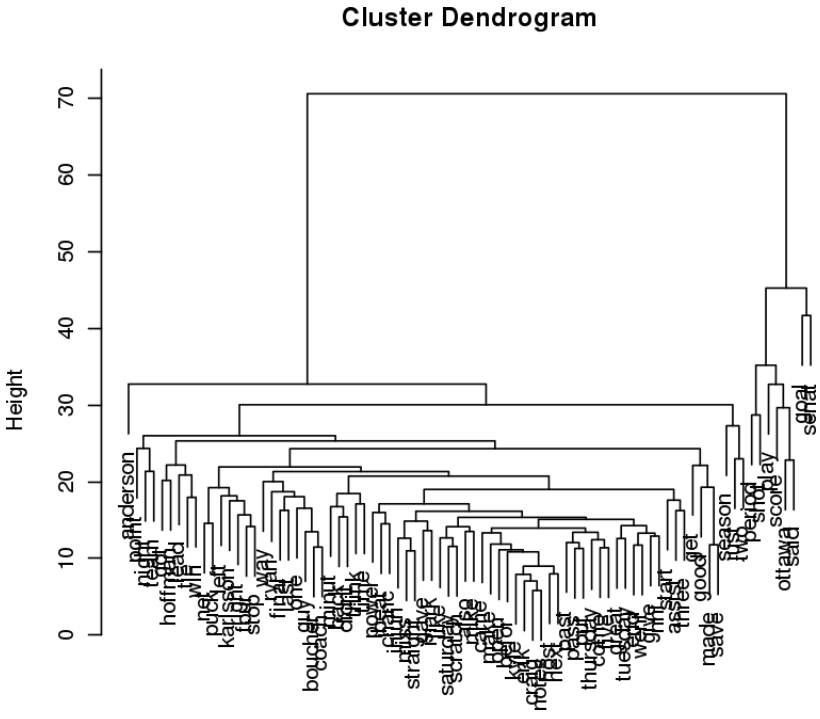
The **document signatures** of the terms have a fair number of non-zero entries, as we would expect since we eliminated sparse terms.

```
# Compute distance matrices
dist_50 = dist(TDM_50.df)
```

```
             also  anderson    assist      back      beat     befor   boucher ...
anderson 28.160256
assist   14.798649 30.594117
back     15.874508 26.095977 18.947295
beat     15.524175 27.313001 18.000000 15.066519
befor    12.529964 30.331502 16.613248 17.464249 14.142136
boucher  14.247807 27.676705 16.431677 16.583124 15.362291 16.613248
came     14.387495 28.879058 16.792856 16.643317 16.552945 13.341664 16.000000 ...
     ... ...       ...       ...       ...       ...       ...       ...       ...
```

With the distance matrix created, the next step is to cluster the terms using the distance matrix.

```
# Build the hc object
hc.50 = hclust(dist_50)

# Plot the dendograms
plot(hc.50)
```



The dendrogram is not easy to read, but we can get the labels directly, if needed.

```
# Build hcd
hcd.50 = as.dendrogram(hc.50)

# labels
labels(hcd.50)
```

```
[1]  'anderson' 'point' 'night' 'team' 'got' 'hoffman' 'lead' 'tie' 'win' 'net' 'puck'
[12] 'left' 'karlsson' 'right' 'four' 'stop' 'way' 'ryan' 'final' 'last' 'one' 'guy'
[23] 'boucher' 'coach' 'minut' 'back' 'didnt' 'think' 'time' 'power' 'beat' 'chanc'
[33] 'injuri' 'miss' 'straight' 'gave' 'mark' 'like' 'saturday' 'scratch' 'also' 'mike'
[43] 'came' 'make' 'open' 'befor' 'kyle' 'erik' 'craig' 'notes' 'host' 'next' 'past'
[54] 'pass' 'put' 'thursday' 'come' 'five' 'great' 'tuesday' 'end' 'went' 'give' 'nhl'
[65] 'start' 'assist' 'three' 'get' 'good' 'made' 'save' 'season' 'just' 'two' 'period'
[76] 'shot' 'play' 'score' 'ottawa' 'said' 'goal' 'senat'
```

Say we are looking for $k = 2$ clusters.

```
result = cutree(hc.50, k=2)

# cluster 1
rownames(TDM_50.df)[result==1]

# cluster 2
rownames(TDM_50.df)[result==2]
```

```
# cluster 1
[1]  'anderson' 'point' 'night' 'team' 'got' 'hoffman' 'lead' 'tie' 'win' 'net' 'puck'
...
[65] 'start' 'assist' 'three' 'get' 'good' 'made' 'save' 'season' 'just' 'two'

# cluster 2
[1] 'period' 'shot' 'play' 'score' 'ottawa' 'said' 'goal' 'senat'
```

The terms found in a cluster are those for whom the document signatures are similar to one another in the corpus.

Using the TDM to find Associations

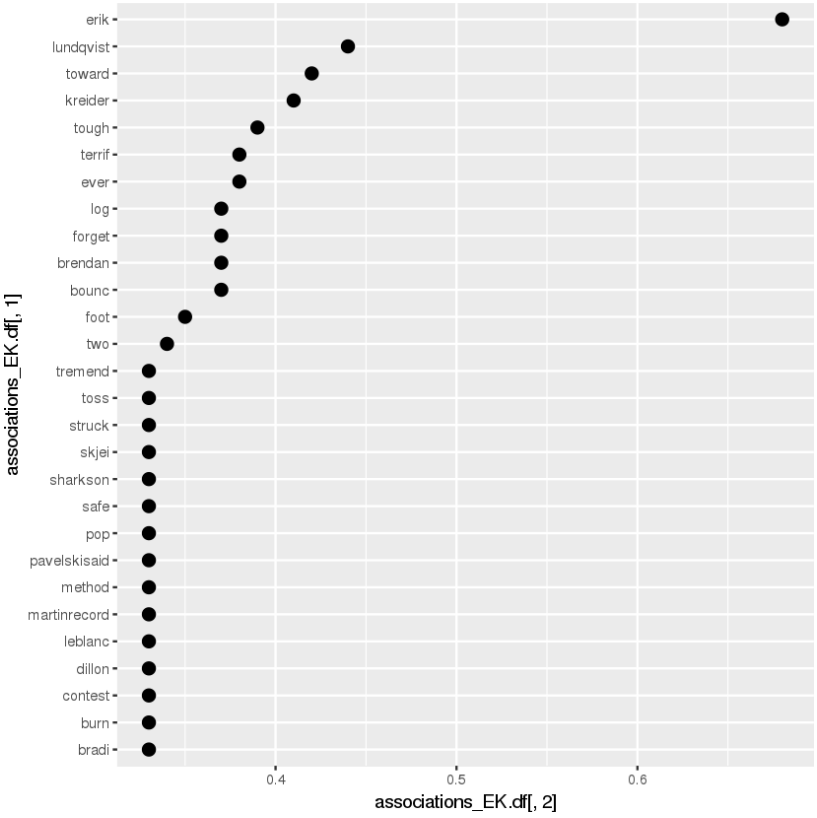
The TDM can also be used to find terms that are **associated with each other** (i.e., terms that appear with each other) across documents. Here we see which other terms are associated with the term 'karlsson',[43] as visualized using a **dotplot**.

43: Former Senators player Erik Karlsson.

```
(associations_EK <- tm::findAssocs(AP.recaps_tdm, "karlsson", 0.33))  # 0.33: lower correlation limit
```

```
$karlsson =
   ever      foot      leblanc   sharkson
   0.38      0.35      0.33      0.33
erik      terrif    two       martinrecord skjei
   0.68      0.38      0.34      0.33      0.33
lundqvist bounc     bradi     method    struck
   0.44      0.37      0.33      0.33      0.33
toward    brendan   burn      pavelskisaid toss
   0.42      0.37      0.33      0.33      0.33
kreider   forget    contest   pop       tremend
   0.41      0.37      0.33      0.33      0.33
tough     log       dillon    safe
   0.39      0.37      0.33      0.33
```

```
associations_EK.df <- qdap::list_vect2df(associations_EK)[,2:3]

library(ggplot2)
ggplot(associations_EK.df,aes(y=associations_EK.df[,1])) +
    geom_point(aes(x=associations_EK.df[,2]), data=associations_EK.df, size = 3)
```



It is not really surprising that the word with the highest co-occurrence frequency is 'erik' (since that is the player's full name), but some of the other counts are perhaps a bit more surprising: Henrik Lundqvist, a fellow Swede, and Chris Kreider played for the New York Rangers that year, a team the Senators faced 10 times in 2016-2017. This may explain why their last names show up in the dotplot.

27.4.6 The Scottish Play

In this section, we analyze the **emotional content** in Shakespeare's *Macbeth*.[44]

44: This example borrows even more heavily than usual from [39].

Sentiment Analysis Workflow

In general, we conduct **term-by-term** sentiment analysis of text as follows:

1. start with text data;
2. un-nest the tokens to produce a first iteration of **tidy text**;
3. clean and process the tidy text as required by the context;

4. join the tidy text to an appropriate **sentiment lexicon**;
5. summarize the tidy text/sentiment lexicon into a first iteration of **summarized text**;
6. clean and analyze the summarized text, and
7. visualize and present the text mining results.

Sentiment Lexicons

Throughout, we will use the sentiment lexicons included with the `tidytext` package: `AFINN`, `nrc`, `bing`, `loughran`.

```
library(tidytext)
library(textdata) # to obtain the Lexicons

AFINN = get_sentiments("afinn") # words on a scale from -5 (negative) to 5 (positive)
BING = get_sentiments("bing") # binary negative/positive
NRC = get_sentiments("nrc") # assigns categories of sentiments (possibly 1+ to a term)
LOUGHRAN = get_sentiments("loughran")
```

We can take a quick look at the 4 lexicons – the first thing to notice is that they do not all contain the same number of observations.

```
str(AFINN)
table(AFINN$value)
head(AFINN)
tail(AFINN)
```

```
spec_tbl_df [2,477 x 2] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ word : chr [1:2477] "abandon" "abandoned" "abandons" "abducted" ...
 $ value: num [1:2477] -2 -2 -2 -2 -2 -2 -3 -3 -3 -3 ...
 - attr(*, "spec")=
  .. cols(
  ..   word = col_character(),
  ..   value = col_double()
  .. )
 - attr(*, "problems")=<externalptr>

 -5  -4  -3  -2  -1   0   1   2   3   4   5
 16  43 264 966 309   1 208 448 172  45   5
```

```
# A tibble: 6 x 2
  word       value
  <chr>      <dbl>
1 abandon       -2
2 abandoned     -2
3 abandons      -2
4 abducted      -2
5 abduction     -2
6 abductions    -2
```

```
# A tibble: 6 x 2
  word     value
  <chr>    <dbl>
1 youthful     2
2 yucky       -2
3 yummy        3
4 zealot      -2
5 zealots     -2
6 zealous      2
```

```
str(BING)
table(BING$sentiment)
head(BING)
tail(BING)
```

```
tibble [6,786 x 2] (S3: tbl_df/tbl/data.frame)
 $ word     : chr [1:6786] "2-faces" "abnormal" "abolish" "abominable" ..
 $ sentiment: chr [1:6786] "negative" "negative" "negative" "negative" ..

negative positive
    4781     2005

# A tibble: 6 x 2
  word       sentiment
  <chr>      <chr>
1 2-faces    negative
2 abnormal   negative
3 abolish    negative
4 abominable negative
5 abominably negative
6 abominate  negative

# A tibble: 6 x 2
  word      sentiment
  <chr>     <chr>
1 zealous   negative
2 zealously negative
3 zenith    positive
4 zest      positive
5 zippy     positive
6 zombie    negative
```

```
str(NRC)
table(NRC$sentiment)
head(NRC)
tail(NRC)
```

```
tibble [13,872 x 2] (S3: tbl_df/tbl/data.frame)
 $ word     : chr [1:13872] "abacus" "abandon" "abandon" "abandon" ...
 $ sentiment: chr [1:13872] "trust" "fear" "negative" "sadness" ...

   anger anticipation      disgust         fear          joy
    1245          837         1056         1474          687
negative     positive      sadness     surprise        trust
    3316         2308         1187          532         1230

# A tibble: 6 x 2
  word      sentiment
  <chr>     <chr>
1 abacus    trust
2 abandon   fear
3 abandon   negative
4 abandon   sadness
5 abandoned anger
6 abandoned fear

# A tibble: 6 x 2
  word    sentiment
  <chr>   <chr>
1 zealous trust
2 zest    anticipation
3 zest    joy
4 zest    positive
5 zest    trust
6 zip     negative
```

```
str(LOUGHRAN)
table(LOUGHRAN$sentiment)
head(LOUGHRAN)
tail(LOUGHRAN)
```

```
tibble [4,150 x 2] (S3: tbl_df/tbl/data.frame)
 $ word     : chr [1:4150] "abandon" "abandoned" "abandoning" "abandonment" ...
 $ sentiment: chr [1:4150] "negative" "negative" "negative" "negative" ...

constraining    litigious     negative     positive  superfluous  uncertainty
         184          904         2355          354           56          297

# A tibble: 6 x 2
  word         sentiment
  <chr>        <chr>
1 abandon      negative
2 abandoned    negative
3 abandoning   negative
4 abandonment  negative
5 abandonments negative
6 abandons     negative

# A tibble: 6 x 2
  word           sentiment
  <chr>          <chr>
1 stratum        superfluous
2 superannuation superfluous
3 theses         superfluous
4 ubiquitous     superfluous
5 wheresoever    superfluous
6 whilst         superfluous
```

At a first glance, it seems that there are more terms in the negative end of the "sentimental spectrum". What kind of an effect might this have on sentiment analysis?[45]

45: Is that the same for every language? French seems to have more room for positive terms, but is that simply confirmation bias at play?

We can also compare how the various lexicons grade specific words – let's take a look at a few possibilities.

```
words = c("abandon","bad","not","cool","egregious","strike")
A.w = AFINN[AFINN$word %in% words,]
A.w$lexicon = "AFINN"
colnames(A.w)[2] = "sentiment"
B.w = BING[BING$word %in% words,]
B.w$lexicon = "BING"
N.w = NRC[NRC$word %in% words,]
N.w$lexicon = "NRC"
L.w = LOUGHRAN[LOUGHRAN$word %in% words,]
L.w$lexicon = "LOUGHRAN"

T.w = rbind(A.w,B.w,N.w,L.w)
dplyr::arrange(T.w,word) |> print(n = Inf)
```

```
# A tibble: 25 x 3
   word      sentiment lexicon
   <chr>     <chr>     <chr>
 1 abandon   -2        AFINN
 2 abandon   fear      NRC
 3 abandon   negative  NRC
 4 abandon   sadness   NRC
 5 abandon   negative  LOUGHRAN
```

```
 6 bad       -3        AFINN
 7 bad       negative  BING
 8 bad       anger     NRC
 9 bad       disgust   NRC
10 bad       fear      NRC
11 bad       negative  NRC
12 bad       sadness   NRC
13 bad       negative  LOUGHRAN
14 cool      1         AFINN
15 cool      positive  BING
16 cool      positive  NRC
17 egregious negative  BING
18 egregious anger     NRC
19 egregious disgust   NRC
20 egregious negative  NRC
21 egregious negative  LOUGHRAN
22 strike    -1        AFINN
23 strike    negative  BING
24 strike    anger     NRC
25 strike    negative  NRC
```

Notes and Comments

- Does it make sense to use a social media lexicon to analyze emotional content in Shakespeare's plays? **Context-specific lexicons** can always be used, instead, but they first need to be **built** (time-consuming) and **validated** (requires domain expertise).
- Beware the no-free lunch theorem: the most suitable lexicon may change from project to project.
- As a rule of thumb, it appears that applying sentiment analysis to any text that is intelligible without a slew of annotations is likely to yield more insight than text that requires annotation, but the latter can still be valuable.
- In general, it seems easier to identify a clear sentiment in a short text than in a long one.
- The **unigram** term "bad" is identified as a negative word, whereas "not" is seen as neutral, but "not bad" would be a mostly positive **bigram**. We have discussed bigrams in Section 27.4.2; there is a lot more to be said on the topic (see Chapter 32).

Matching Sentiments to Words

Let's take a quick look at how we can set-up the matching between sentiments and terms using a lexicon. For the purposes of this example, we will use the NRC lexicon, together with *A Midsummer Night's Dream*, one of the more whimsical of Shakespeare's comedies. We start by creating a custom lexicon for the works of Shakespeare at the Gutenberg Project.

```
word = c("etext", "copyright", "implications", "electronic", "version", "william",
         "shakespeare",  "inc", "gutenberg", "electronic", "machine", "distributed",
         "commercially", "commercial", "distribution", "download", "shareware")
```

```
lexicon = rep("custom",17)
custom = data.frame(word,lexicon)
stop_words_custom_gut = rbind(tidytext::stop_words,custom)
```

Next, we extract the play and put it into a tidy dataset.

```
my_mirror = "http://mirror.csclub.uwaterloo.ca/gutenberg/"
msnd <- gutenbergr::gutenberg_download(c(1514),my_mirror)

tidy_msnd <- msnd |>
  tidytext::unnest_tokens(word,text) |>
  dplyr::mutate(word = stringr::str_extract(word,"[a-z']+")) |> # removing odd encodings
  dplyr::anti_join(stop_words_custom_gut) |>  # removing the non-play terms
  na.omit() # remove NAs
```

Now, let's extract the `surprise` words (and attendent frequencies) from *A Midsummer Night's Dream* (according to the NRC lexicon):

```
nrc_surprise <- NRC |>
  dplyr::filter(sentiment == "surprise")

tidy_msnd |>
  dplyr::inner_join(nrc_surprise) |>
  dplyr::count(word, sort = TRUE) |> print(n = Inf)
```

```
# A tibble: 58 x 2
   word          n
   <chr>     <int>
 1 sweet        48
 2 art          16
 3 leave        15
 4 death        14
 5 pray         13
 6 break         7
 7 lovely        7
 8 marry         7
 9 youth         7
10 catch         5
11 wild          5
12 jest          4
13 kiss          4
14 lose          4
15 spirits       4
16 cheer         3
17 fright        3
18 hope          3
19 judgment      3
20 luck          3
21 perchance     3
22 teach         3
23 chance        2
24 mouth         2
25 musical       2
26 revenge       2
27 stealth       2
28 sun           2
29 trip          2
30 worm          2
31 advance       1
32 angel         1
33 ceremony      1
34 conjure       1
35 frantic       1
36 gift          1
37 illusion      1
38 laughter      1
39 lightning     1
40 marvel        1
41 merriment     1
42 murder        1
43 palpable      1
44 precious      1
45 saint         1
46 scare         1
47 screech       1
48 senseless     1
49 shot          1
50 shout         1
51 shriek        1
52 slip          1
53 smile         1
54 sunny         1
55 tempest       1
56 thief         1
57 tickle        1
58 vanished      1
```

We can do the same for the `anger` terms.

```
nrc_anger <- NRC |>
  dplyr::filter(sentiment == "anger")

tidy_msnd |>
  dplyr::inner_join(nrc_anger) |>
  dplyr::count(word, sort = TRUE) |> print(n = Inf)
```

```
# A tibble: 133 x 2
    word              n
    <chr>         <int>
  1 fear             17
  2 death            14
  3 bear             10
  4 hate              9
  5 scorn             9
  6 lie               7
  7 youth             7
  8 mistress          6
  9 hell              5
 10 spite             5
 11 strike            5
 12 cross             4
 13 derision          4
 14 force             4
 15 lose              4
 16 steal             4
 17 words             4
 18 adder             3
 19 beast             3
 20 bellows           3
 21 bloody            3
 22 cruel             3
 23 fierce            3
 24 hateful           3
 25 honest            3
 26 ill               3
 27 injury            3
 28 offend            3
 29 wound             3
 30 angry             2
 31 bully             2
 32 confusion         2
 33 curse             2
 34 dame              2
 35 darkness          2
 36 delay             2
 37 deny              2
 38 deserve           2
 39 discord           2
 40 dreadful          2
 41 fight             2
 42 grim              2
 43 hatred            2
 44 hurt              2
 45 jealous           2
 46 jealousy          2
 47 liquor            2
 48 loath             2
 49 loathe            2
 50 lying             2
 51 mad               2
 52 murderer          2
 53 musical           2
 54 rage              2
 55 revenge           2
 56 tyrant            2
 57 wasted            2
 58 anger             1
 59 anguish           1
 60 argument          1
 61 bark              1
 62 battle            1
 63 bee               1
 64 burial            1
 65 complaint         1
 66 cruelty           1
 67 cur               1
 68 damn              1
 69 deceive           1
 70 defect            1
 71 desert            1
 72 despise           1
 73 detest            1
 74 disgrace          1
 75 disobedience      1
 76 disparage         1
 77 dissension        1
 78 distracted        1
 79 elf               1
 80 enmity            1
 81 exile             1
 82 fee               1
 83 foe               1
 84 foul              1
 85 fury              1
 86 gore              1
 87 guilty            1
 88 harbinger         1
 89 harshness         1
 90 hit               1
 91 hot               1
 92 hunting           1
 93 injurious         1
 94 insufficiency     1
 95 killing           1
 96 lightning         1
 97 lunatic           1
 98 madman            1
 99 mighty            1
100 miserable         1
101 mislead           1
102 moral             1
103 murder            1
104 odious            1
105 offended          1
106 pare              1
107 prison            1
108 prosecute         1
109 raging            1
110 rail              1
111 riot              1
112 scar              1
113 scare             1
114 senseless         1
115 shiver            1
116 shot              1
117 shout             1
118 shriek            1
119 shun              1
120 sinister          1
121 slay              1
122 stone             1
123 strife            1
124 tempest           1
125 thief             1
126 throttle          1
127 torment           1
128 ungrateful        1
129 unkind            1
130 warrior           1
131 whip              1
132 wrath             1
133 wretch            1
```

Note that there are overlaps: "revenge", for instance, is in both collections. In total, there are 286 occurrences of `anger` terms in the cleaned up text data, and 225 occurrences of `surprise` terms. Does this fit with the nature of the play?

term-by-term Sentiment Analysis of Macbeth

Instead of finding words that express specific sentiments, we are going to compute a **score** for various sections of *Macbeth*.

First, we load a processed version of the text.

```
macbeth = read.csv("Macbeth.csv", header=TRUE, sep=",",
                   stringsAsFactors=FALSE)
str(macbeth)
```

```
'data.frame': 15221 obs. of  6 variables:
 $ Act       : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Scene     : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Speaker   : chr  "First Witch" "First Witch" "Second Witch" "Second Witch" ...
 $ Text      : chr  "When shall we three meet again" "In thunder, lightning, or in rain?" ...
 $ Scene_Line: int  1 2 3 4 5 6 7 8 9 10 ...
 $ Play_Line : int  1 2 3 4 5 6 7 8 9 10 ...
```

The `Act` and `Scene` variables could be combined to provide an increasing identifier for the play's sections. For the purpose of this example, we only want to keep information on the text, the line number, and the section.

```
macbeth$section = macbeth$Act*10 + macbeth$Scene
macbeth <- macbeth |>
  dplyr::select(c("Text","Play_Line","section"))
head(macbeth)
```

```
                             Text Play_Line section
1     When shall we three meet again         1      11
2 In thunder, lightning, or in rain?         2      11
3        When the hurlyburly's done,         3      11
4    When the battle's lost and won.         4      11
5   That will be ere the set of sun.         5      11
6                   Where the place?         6      11
```

Next, we unnest the tokens into a tidy format, using word as the basic unit.

```
tidy_macbeth <- macbeth |> tidytext::unnest_tokens(word, Text)
head(tidy_macbeth,20)
```

```
   Play_Line section         word
1          1      11         when
2          1      11        shall
3          1      11           we
4          1      11        three
5          1      11         meet
6          1      11        again
7          2      11           in
8          2      11      thunder
9          2      11    lightning
```

```
10         2      11           or
11         2      11           in
12         2      11         rain
13         3      11         when
14         3      11          the
15         3      11 hurlyburly's
16         3      11         done
17         4      11         when
18         4      11          the
19         4      11     battle's
20         4      11         lost
```

At this point, we get a sentiment score for each word using the BING lexicon (words that don't appear in the lexicon are assumed to be neutral).

```
macbeth_SA <- tidy_macbeth |> dplyr::inner_join(BING)
head(macbeth_SA)
```

```
  Play_Line section word sentiment
1         4      11 lost  negative
2         4      11  won  positive
3        12      11 fair  positive
4        12      11 foul  negative
5        12      11 foul  negative
6        12      11 fair  positive
```

Next, we count the `positive` and `negative` words in each grouping of $L = 30$ lines of text, say.

```
macbeth_SA <- tidy_macbeth |> dplyr::inner_join(BING) |>
  dplyr::count(index = Play_Line %/% 30, sentiment)
head(macbeth_SA)
```

```
  index sentiment  n
1     0  negative 11
2     0  positive 10
3     1  negative 12
4     1  positive 10
5     2  negative 17
6     2  positive 11
```

The counts are stored in the variable `n`. We reshape the tibble into a **tidy** dataset, one for which each column hosts 1 variable, and each row, 1 observation.

```
macbeth_SA <- tidy_macbeth |> dplyr::inner_join(BING) |>
  dplyr::count(index = Play_Line %/% 30, sentiment) |>
  tidyr::spread(sentiment, n, fill = 0)
head(macbeth_SA)
```

```
  index negative positive
1     0       11       10
2     1       12       10
3     2       17       11
4     3        7        2
5     4        7       14
6     5        8        9
```

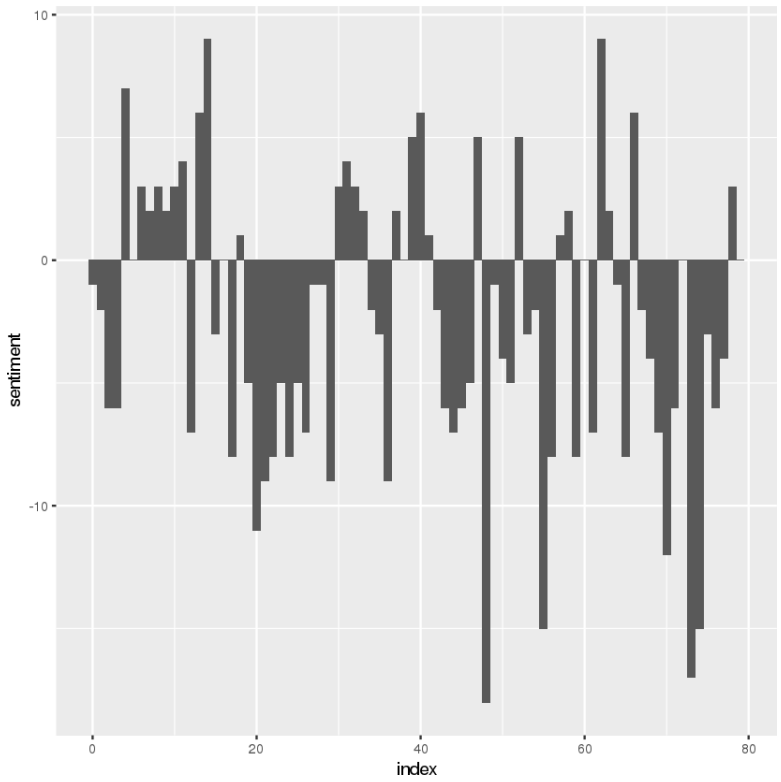
Finally, we compute the **overall sentiment** for each block of lines as the difference between its positive and negative term counts.

```
macbeth_SA <- tidy_macbeth |> dplyr::inner_join(BING) |>
  dplyr::count(index = Play_Line %/% 30, sentiment) |>
  tidyr::spread(sentiment, n, fill = 0) |>
  dplyr::mutate(sentiment = positive - negative)
head(macbeth_SA)
```

```
  index negative positive sentiment
1     0       11       10        -1
2     1       12       10        -2
3     2       17       11        -6
4     3        7        2        -5
5     4        7       14         7
6     5        8        9         1
```

That's it! – although it might be more meaningful to plot the results.

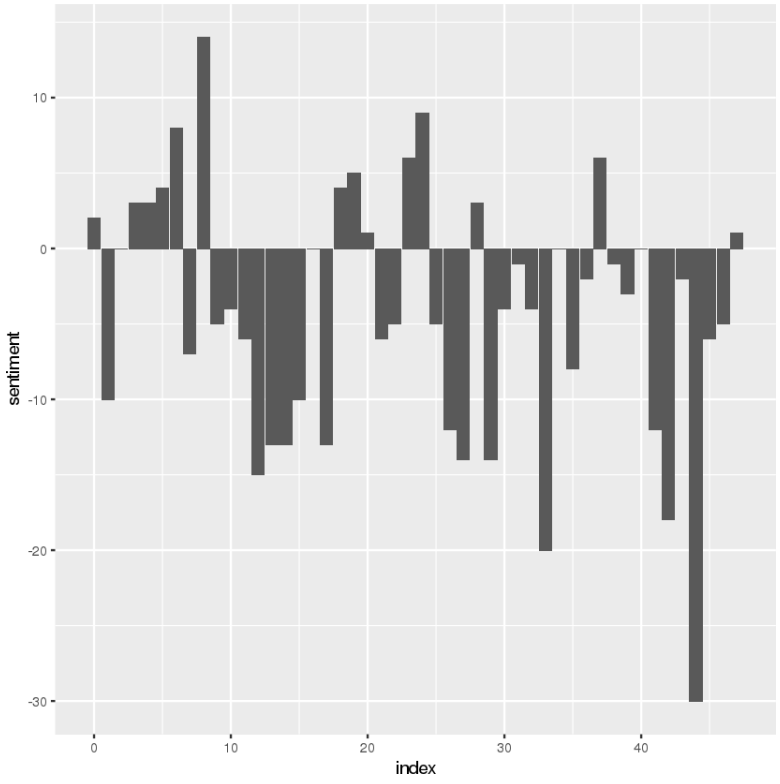
```
library(ggplot2)
ggplot(macbeth_SA, aes(index, sentiment)) + geom_col()
```



The overall picture seems to be somewhat negative – but is that surprising? *Macbeth* is a tragedy, after all, arguably Shakespeare's darkest. Perhaps what we're seeing is an artifact of the way we have **blocked** (grouped) the play, or of the length of the blocks, or even of the sentiment lexicon that we've elected to use. We look into this a little bit more.

Smaller Number of Blocks What happens if we use $L = 50$ instead of $L = 30$?

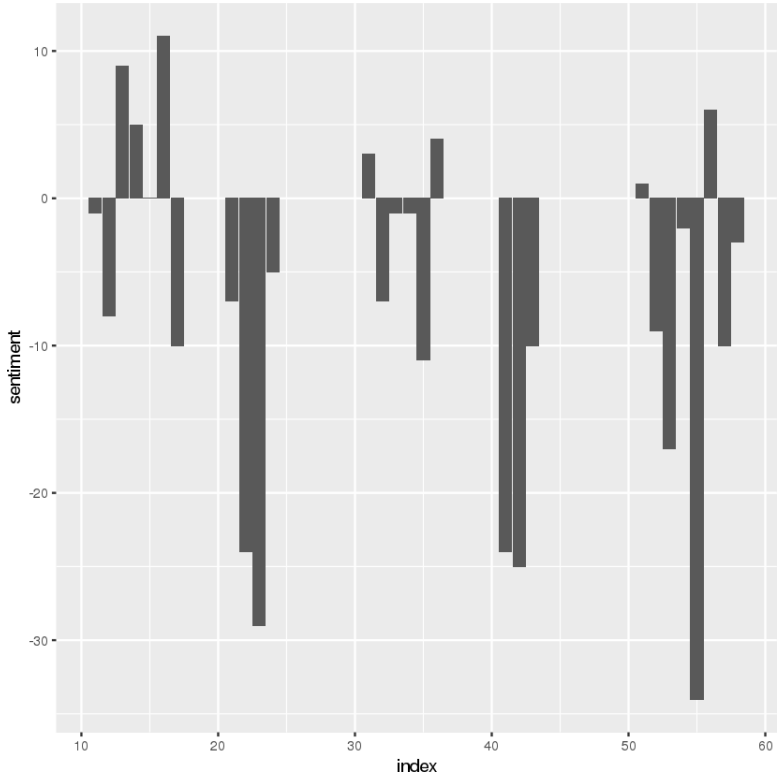
```
macbeth_SA <- tidy_macbeth |> dplyr::inner_join(BING) |>
  dplyr::count(index = Play_Line %/% 50, sentiment) |>
  tidyr::spread(sentiment, n, fill = 0) |>
  dplyr::mutate(sentiment = positive - negative)

ggplot(macbeth_SA, aes(index, sentiment)) + geom_col()
```



There isn't much of a difference.

Different Blocking Mechanism We could use `Act` and `Scene` as separation instead of an arbitrary number of lines L.

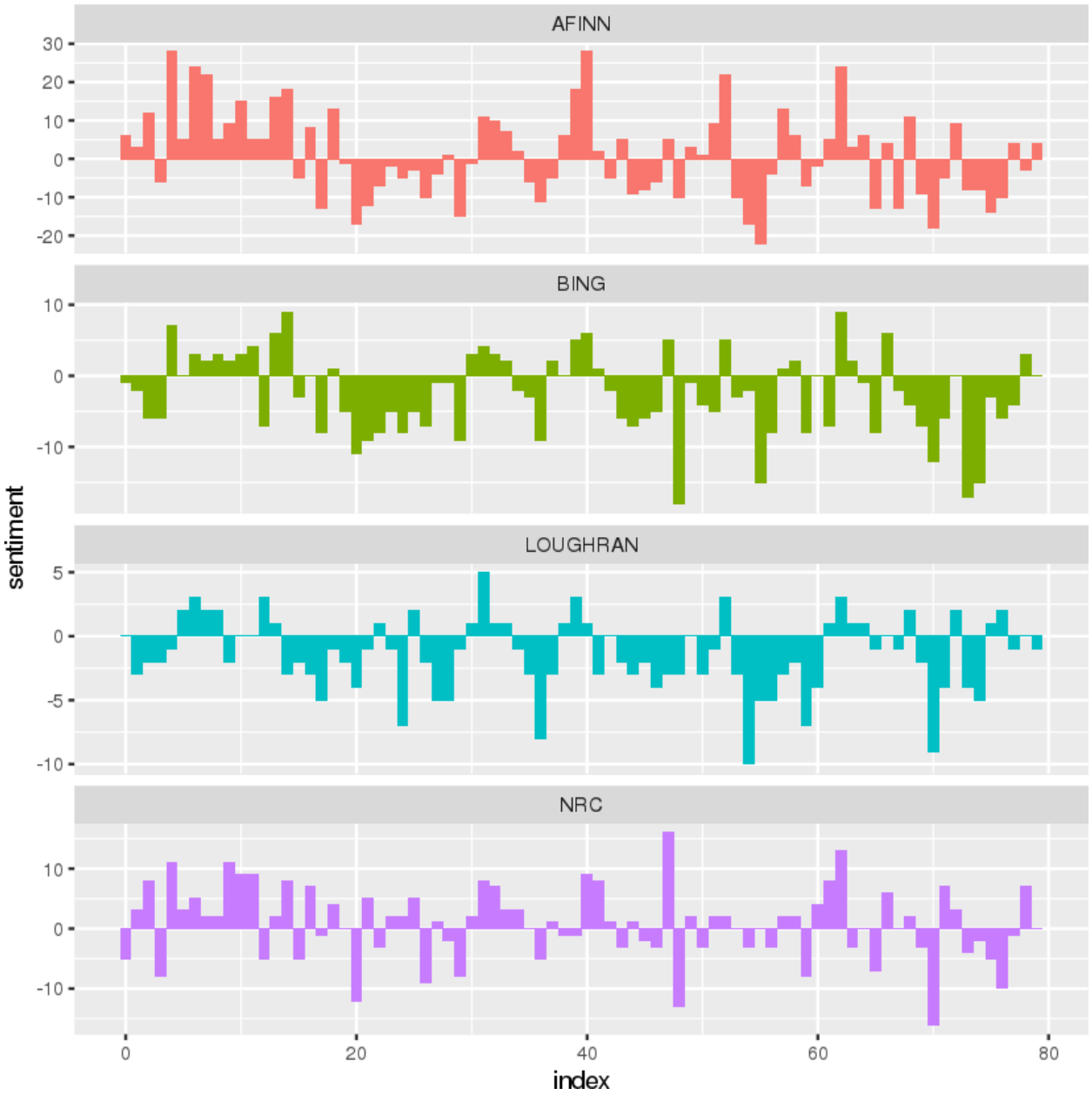
```
macbeth_SA <- tidy_macbeth |> dplyr::inner_join(BING) |>
  dplyr::count(index = section, sentiment) |>
  tidyr::spread(sentiment, n, fill = 0) |>
  dplyr::mutate(sentiment = positive - negative)

ggplot(macbeth_SA, aes(index, sentiment)) + geom_col()
```



Acts II, IV, and V are pretty bleak, seems like...

Different Lexicons We go back to $L = 30$ and run term-by-term sentiment analysis for the four lexicons.

```
afinn_macbeth <- tidy_macbeth |> dplyr::inner_join(AFINN) |>
  dplyr::group_by(index = Play_Line %/% 30) |>
  dplyr::summarise(sentiment = sum(value)) |> dplyr::mutate(method = "AFINN")

bing_nrc_loughran_macbeth <- dplyr::bind_rows(
  tidy_macbeth |>
    dplyr::inner_join(BING) |>
    dplyr::mutate(method = "BING"),
  tidy_macbeth |>
    dplyr::inner_join(NRC |> dplyr::filter(sentiment %in% c("positive","negative"))) |>
    dplyr::mutate(method = "NRC"),
  tidy_macbeth |>
    dplyr::inner_join(LOUGHRAN |> dplyr::filter(sentiment %in% c("positive","negative"))) |>
    dplyr::mutate(method = "LOUGHRAN")) |>
  dplyr::count(method, index = Play_Line %/% 30, sentiment) |>
  tidyr::spread(sentiment, n, fill = 0) |>
  dplyr::mutate(sentiment = positive - negative)

library(ggplot2)
dplyr::bind_rows(afinn_macbeth, bing_nrc_loughran_macbeth) |>
  ggplot(aes(index, sentiment, fill = method)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~method, ncol = 1, scales = "free_y")
```



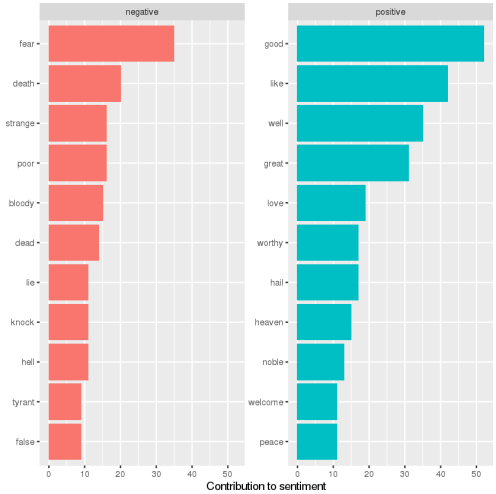
With BING and LOUGHRAN, the "tragedy" of *Macbeth* is "preserved", but that pattern is not as obvious with AFINN and NRC (although there is plenty of negative sentiment in those two as well).

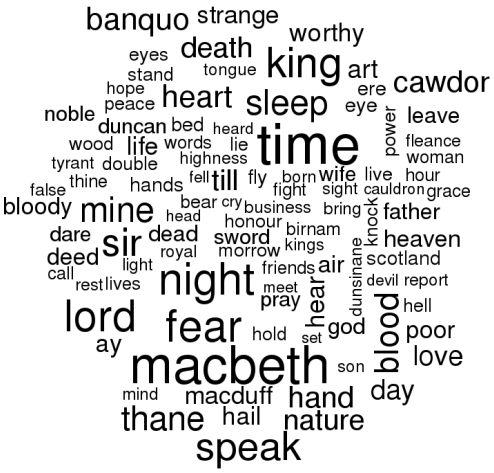
Perhaps we should question the wisdom of using modern lexicons on 450 year old plays?

Text Visualizations Finally, we can also look at how often specific words contribute to positive and negative sentiments in the text of the play, using the BING lexicon.

```
bing_word_counts <- tidy_macbeth |>
  dplyr::inner_join(get_sentiments("bing")) |>
  dplyr::count(word, sentiment, sort = TRUE) |> dplyr::ungroup()
```

```
library(ggplot2)

#bar charts
bing_word_counts |>
  dplyr::group_by(sentiment) |> # will create 2 graphs
  dplyr::top_n(10) |> # pick only the top 10 in each category
  dplyr::ungroup() |> # required to avoid a warning message below
  dplyr::mutate(word = reorder(word, n)) |>
  ggplot(aes(word, n, fill = sentiment)) + # plot a bar chart of word count
  geom_col(show.legend = FALSE) +
  facet_wrap(~sentiment, scales = "free_y") + # there will 2 such bar charts
  labs(y = "Contribution to sentiment",x = NULL) +
  coord_flip() # horizontal bar charts

# wordcloud
word = c("thou", "thy", "thee", "tis", "hath")
lexicon = rep("custom",5)
custom2 = data.frame(word,lexicon)
stop_words_custom_macbeth = rbind(tidytext::stop_words,custom2)
tidy_macbeth |> dplyr::anti_join(stop_words_custom_macbeth) |>
  dplyr::count(word) |>
  with(wordcloud::wordcloud(word, n, max.words = 100))

# comparison cloud
tidy_macbeth |> dplyr::inner_join(BING) |>
  dplyr::count(word, sentiment, sort = TRUE) |>  # counting words for the whole play
  reshape2::acast(word ~ sentiment, value.var = "n", fill = 0) |> # reshaping as a matrix
  wordcloud::comparison.cloud(colors = c("#660000", "#000066"), max.words = 100)
```





Nothing should jump at us as being amiss – which is no guarantee that there's no problem, but it's at least a good sign.

Notes and Comments We see that the choice of lexicon and of the blocking window may have an impact on the sentiment analysis outcome. This is aligned with data anlaysis as we have seen it so far: it's easy to run a sentiment analysis (a few lines of code at most), but it's difficult to pick (or build) the right window and the right lexicon.

27.4.7 Regular Expressions

(This section is a repeat of 16.6.4)

Regular expressions can be used to achieve to extract relevant information from reams of data. Among this mostly unstructured data lurk **systematic elements**, which can be used to help the automation process, especially if quantitative methods are eventually going to be applied to the scraped data.

Systematic structures include numbers, names (countries, etc.), addresses (mailing, e-mailing, URLs, etc.), specific character strings, etc. Regular expressions (**regexps**) are abstract sequences of strings that match concrete recurring patterns in text; they allow for the systematic extraction of the information components from plain text, HTML, and XML.

The examples in this section are based on [23].

Initializing the Environment

The `Python` module for regular expressions is `re`.

```
import re
```

Let us take a quick look at some basics, through the `re` method `match()`. We can try to match a pattern from the beginning of a string, as below:

```
re.match('super','supercalifragilisticexpialidocious')
```

```
<re.Match object; span=(0, 5), match='super'>
```

No such match occurs in the following chunk of code, however.

```
re.match('super','Supercalifragilisticexpialidocious')
```

The regular expression pattern (more on this in a moment) for "word" is `\w+`. The following bit of code would match the first word in a string:

```
w_regex = '\w+'
re.match(w_regex,'Hello World!')
```

```
<re.Match object; span=(0, 5), match='Hello'>
```

Common Regular Expression Patterns

A **regular expression pattern** is a short form used to indicate a type of (sub)string:

- `\w+`: word
- `\d`: digit
- `\s`: space
- `.`: wildcard
- `+` or `*`: greedy match
- `\W`: not word
- `\D`: not digit
- `\S`: not space
- `[a-z]`: lower case group
- `[A-Z]`: upper case group

There are a few `re` functions which, combined with regexps, can make it easier to extract information from large, unstructured text documents:

- `split()`: splits a string on a regexp;
- `findall()`: finds all substrings matching a regexp in a string;
- `search()`: searches for a regexp in a string, and
- `match()`: matches an entire string based on a regexp

Each of these functions takes two arguments: a **regexp** (first) and a **string** (second). For instance, we can split a string on the spaces (and remove them):

```
re.split('\s+','Can  you do the split?')
```

```
['Can', 'you', 'do', 'the', 'split?']
```

The `\` in the regexp above is crucial. The following code splits the sentence on the `s` (and removes them):

```
re.split('s+','Can  you do the split?')
```

```
['Can  you do the ', 'plit?']
```

We can also split on single spaces and remove them:

```
re.split('\s','Can  you do the split?')
```

```
['Can', '', 'you', 'do', 'the', 'split?']
```

Alternatively, we can also split on the words and remove them:

```
re.split('\w+','Can  you do the split?')
```

```
['', '  ', ' ', ' ', ' ', '?']
```

Or better yet, split on the non-words and remove them:

```
re.split('\W+','Can you do the split?')
```

```
['Can', 'you', 'do', 'the', 'split', '']
```

Let us take some time to study a silly sentence, saved as a string.

```
test_string = 'Oh they built the built the ship Titanic.
  It was a mistake. It cost more than 1.5 million dollars.
  Never again!'
test_string
```

```
'Oh they built the built the ship Titanic. It was a mistake.
It cost more than 1.5 million dollars. Never again!'
```

In English, only three characters can end a sentence: `.`, `?`, `!`.[46] We create a regexp group (more on those in a moment) as follows:[47]

```
sent_ends = r"[.?!]"
```

We could then split the string into its constituent sentences:

```
print(re.split(sent_ends,test_string))
```

```
['Oh they built the built the ship Titanic', ' It was a mistake',
 ' It cost more than 1', '5 million dollars', ' Never again', '']
```

If we wanted to know how many such sentences there were, we simply use the `len()` function:

```
print(len(re.split(sent_ends,test_string)))
```

```
6
```

The regexp range consisting of words with an uppercase initial letter is easy to build:

```
cap_words = r"[A-Z]\w+" # Upper case characters
```

We can find all such words (and how many there are in the string) through:

```
print(re.findall(cap_words,test_string))
print(len(re.findall(cap_words,test_string)))
```

```
['Oh', 'Titanic', 'It', 'It', 'Never']
5
```

The regexp for spaces is:

46: Apparently, nobody's heard of the interrobang...

47: In `Python`, regular expression patterns must be prefixed with an `r` to differentiate between the **raw string** and the **string's interpretation**.

```
spaces = r"\s+" # spaces
```

We can then split the string on spaces, and count the number of **tokens** (see Chapter 27, *Text Analysis and Text Mining*):

```
print(re.split(spaces,test_string))
print(len(re.split(spaces,test_string)))
```

```
['Oh', 'they', 'built', 'the', 'built', 'the', 'ship', 'Titanic.',
 'It', 'was', 'a', 'mistake.', 'It', 'cost', 'more', 'than', '1.5',
 'million', 'dollars.', 'Never', 'again!']
21
```

The regexp for numbers (contiguous strings of digits) is:

```
numbers = r"\d+"
```

We can find all the numeric characters using:

```
print(re.findall(numbers,test_string))
print(len(re.findall(numbers,test_string)))
```

```
['1', '5']
2
```

The main difference between `search()` and `match()` is that `match()` tries to match from the beginning of a string, whereas `search()` looks for a match anywhere in the string.

Regular Expressions Groups '( )' and Ranges '[ ]' With OR '|'

We can create more complicated regexps using **groups**, **ranges**, and/or "or" statements:

- `[a-zA-Z]+`: an unlimited number of lower and upper case English/French (unaccented) letters;
- `[0-9]`: the digits from 0 to 9;
- `[a-zA-Z'\.\-]+`: any combination of lower and upper case English/French (unaccented) letters, `'`, `.`, and `-`;
- `(a-z)`: the characters `a`, `-`, and `z`;
- `(\s+|,)`: any number of spaces, or a comma;
- `(\d+|\w+)`: words or numerics.

For instance, consider the following text string and regexps groups:

```
text = 'On the 1st day of xmas, my boat sank.'
numbers_or_words = r"(\d+|\w+)"
spaces_or_commas = r"(\s+|,)"
```

This next chunk of code does exactly what one would expect:

```
print(re.findall(numbers_or_words,text))
```

```
['On', 'the', '1', 'st', 'day', 'of', 'xmas', 'my', 'boat', 'sank']
```

What about this one?

```
print(re.findall(spaces_or_commas,text))
```

```
[' ', ' ', ' ', ' ', ' ', ',', ' ', ' ', ' ']
```

27.4.8 Movie Reviews

> "This is a failure of epic proportions. You've got to be a genius to make a movie this bad." (J. Seigel's review of The *Bonfire of the Vanities*)

In this section, we will re-visit sentiment analysis, this time using `Python` and the Natural Language Toolkit (NLTK).[48]

48: See Chapter 32 for more examples of NLTK in action.

Our goal is to develop a sentiment analysis model for movie reviews. The dataset contains 50,000 movie reviews labeled as either **positive** or **negative**. With an accurate sentiment model, we'll have the ability to automatically classify new reviews in order to aggregate review data.

Dataset Information

The `Large Movie Review Dataset v1.0` is described in [36] – the details below are taken verbatim from the same source (the data is also available in the `R` package `textdata`, in the object `dataset_imdb`).

Overview This dataset contains movie reviews along with their associated binary sentiment polarity labels. It is intended to serve as a benchmark for sentiment classification. This document outlines how the dataset was gathered, and how to use the files provided.

Dataset The core dataset contains 50,000 reviews split evenly into 25K train and 25K test sets. The overall distribution of labels is balanced (25K pos and 25K neg). We also include an additional 50,000 unlabeled documents for unsupervised learning.

In the entire collection, no more than 30 reviews are allowed for any given movie because reviews for the same movie tend to have correlated ratings. Further, the training and test sets contain a disjoint set of movies, so no significant performance is obtained by memorizing movie-unique terms and their associated with observed labels. In the labeled train/test sets, a negative review has a score ≤ 4 out of 10, and a positive review has a score ≥ 7 out of 10. Thus reviews with more neutral ratings are not included in the train/test sets. In the unsupervised set, reviews of any rating are included and there are an even number of reviews > 5 and ≤ 5.

Files There are two top-level directories [`train/`, `test/`] corresponding to the training and test sets. Each contains [`pos/`, `neg/`] directories for the reviews with binary labels positive and negative. Within these directories, reviews are stored in text files named following the convention [`[id]_[rating].txt`] where `[id]` is a unique id and `[rating]` is the star rating for that review on a 1-10 scale. For example, the file [`test/pos/200_8.txt`] is the text for a positive-labeled test set example with unique id 200 and star rating 8/10 from IMDb. The [`train/unsup/`] directory has 0 for all ratings because the ratings are omitted for this portion of the dataset.

Preamble

We first need to import the appropriate Python modules. For this exercise, we'll use NLTK.

The `stemmer()` and `tokenize()` functions are used for text processing. The `vader` lexicon is used to analyze the **intensity** of the sentiments.

```
import nltk
from nltk.classify import NaiveBayesClassifier
from nltk.corpus import movie_reviews
from nltk.sentiment import SentimentAnalyzer
from nltk.sentiment.util import mark_negation, extract_unigram_feats
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem.lancaster import LancasterStemmer
stemmer = LancasterStemmer()

import glob
data_path = 'Data/aclImdb/'

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
from nltk.sentiment.vader import SentimentIntensityAnalyzer

nltk.download('all')
```

Data Preparation

The reviews are individually stored in text files, and there are four folders for every combination of training/test and positive/negative.

```
train_docs = []
train_labels = []

pos_file_names = glob.glob('{}train/pos/*.txt'.format(data_path))
for file_name in pos_file_names:
    train_docs.append(open(file_name).read())
    train_labels.append(1)
```

```
neg_file_names = glob.glob('{}train/neg/*.txt'.format(data_path))
for file_name in neg_file_names:
    train_docs.append(open(file_name).read())
    train_labels.append(0)
```

The names of the positive reviews are found in an array, just as the names of the negative reviews are.

```
pos_file_names
```

```
['Data/aclImdb/train/pos/0_9.txt',
 'Data/aclImdb/train/pos/10000_8.txt',
 'Data/aclImdb/train/pos/10001_10.txt',
 'Data/aclImdb/train/pos/10002_7.txt',
 ...]
```

We read-in a (random) sample negative review, for the movie *Haunted Boat*, whose file number is `3446_1.txt`. The '_1' in the file title lets us know that this is a 1-star review. Does the text support the rating?

```
sample_text_neg = open('{}train/neg/3446_1.txt'.format(data_path)).read()
print(sample_text_neg)
```

```
This film on paper looked like it could possibly be good, after watching though i realised
that this film was completely terrible!! The plot has no meaning, and i think i counted the
best part of 5000 cut scenes each one making the film more annoying boring and ridiculous. I
watched this late night pitch black no noise at all just to add to the SCARINESS of it but
the truth is the only thing that scared me was the music, what they would call tragic music,
they play opera i mean be serious!! This film sums up all of what is not good about this type
of film. To be honest ill say no more but watch at your own risk this film is just complete
rubbish, ENJOY!!
```

Next, we read-in a positive review, now, for a movie called *The Night Listener*, whose file number is `10015_8.txt`. This review is supposed to be an 8-star review – does the text support the rating?

```
sample_text_pos = open('{}train/pos/10015_8.txt'.format(data_path)).read()
print(sample_text_pos)
```

```
Popular radio storyteller Gabriel No one(Robin Williams,scraggy and speaking in hushed,
hypnotic tones) becomes acquainted and friends with a fourteen-year-old boy from Wisconsin
named Pete Logand(Rory Culkin),who has written a book detailing sexual abuse from his parents.
To boot,Pete has AIDS and this compels Gabriel further still,since his partner Jess(Bobby
Cannavale, good)happens to be a survivor of HIV himself. <br /><br />He also acquaints himself
with Pete's guardian,a woman named Donna(Toni Collette, brilliant!)and when Gabriel decides he
wants to meet and talk to the two of them in person and goes to Wisconsin, [...]
```

Bag-of-Words Processing

We will be using a BoW model, so let's explore how we could tokenize (that is, separate) the text into words (the **tokens**).

First, to split a review into sentences we can use the standard `sent_tokenize()` function from NLTK. For instance, the following piece of code will extract the 4th sentence that the tokenizer recognizes (in `Python`, indexing starts with 0).

```
sample_sent = sent_tokenize(sample_text_neg)[3]
print(sample_sent)
```

```
This film sums up all of what is not good about this type of film.
```

We can also try the `word_tokenize()` function to split into words, and do a stemming operation (finding the **roots**) to normalize word forms. The following code will stem all the words in the `sample_sent` sentence from above.

```
sample_words = [ stemmer.stem(word)
                 for word in word_tokenize(sample_sent) ]
print(sample_words)
```

```
['thi', 'film', 'sum', 'up', 'al', 'of', 'what', 'is', 'not',
'good', 'about', 'thi', 'typ', 'of', 'film', '.']
```

One serious problem with a bag of words approach, especially for sentiment analysis, is that the presence of negative/positive words does not imply negative/positive sentiment if the words are negated in the sentence (e.g., "not bad" actually means "good" even though in general an occurrence of "bad" means "bad").

NLTK includes the function `mark_negation()` which takes a tokenized sentence and marks negated words with a '`_NEG`' suffix. Specifically, it marks all words that come after a negation word and before the next punctuation mark. Now 'good' becomes the word '`good_NEG`' so a BoW model can pick up on the **context** of the word.[49]

49: More on this topic in Chapter 32.

```
mark_negation(sample_words)
```

```
['thi',
 'film',
 'sum',
 'up',
 'al',
 'of',
 'what',
 'is',
 'not',
 'good_NEG',
 'about_NEG',
 'thi_NEG',
 'typ_NEG',
 'of_NEG',
 'film_NEG',
 '.']
```

Here is the complete tokenizer function:

1. it tokenizes the text into sentences;
2. for each tokenized sentence, it tokenizes it into words;
3. it keeps only those words of length ≥ 2;
4. it stems the words to only retain the roots, and
5. it marks the negation of certain words.

```
def tokenizer(text):
    sents = sent_tokenize(text)
    tokens = []
    for sent in sents:
        words = word_tokenize(sent)
        words = [ word for word in words if len(word) >= 2 ]
        words = [ stemmer.stem(word) for word in words ]
        words = mark_negation(words)
        tokens += words
    return tokens
```

Now that we have a tokenizer, we can use standard feature extraction methods to get feature vectors for each document.

We use the `scikit-learn` module for the rest of the feature extraction and training; it contains `TfidfVectorizer` class which allows us to define a custom tokenizer and returns a `TFIDF` matrix.[50]

50: We use this class to convert all of the training documents to DTM feature vectors, using the tokenizer defined above (this step can take a few minutes to run).

```
vectorizer = TfidfVectorizer(min_df=1, tokenizer=tokenizer)
train_matrix = vectorizer.fit_transform(train_docs)
```

We can explore the matrix to get an idea of what it contains. It should contain 25,000 documents (as per the introduction), but how many features have been retained?

```
train_matrix.shape
```

```
(25000, 105350)
```

A fair number, as it happens: 25,000 documents and 105,494 features. We can also find the non-zero entries among a subset of the DTM matrix, but that doesn't give us much information at this stage (it will only print the non-zero entries, but we don't know what the features are).

```
print(train_matrix[0:9,0:9])
```

```
  (0, 2) 0.0553631491077
  (1, 3) 0.0250716608331
  (2, 2) 0.0341384144135
  (4, 2) 0.100478224693
  (7, 2) 0.141603801927
  (7, 3) 0.0266220723609
```

Multinomial Naïve Bayes

Multinomial naïve Bayes (`MultinomialNB()`) is one of various classification models in `scikit-learn` (if we wanted to find the best possible classifier, we'd have to try some of the others, but at this stage we just want to show you how the sentiment analysis works).

The `fit` function takes the feature matrix as well as the vector of labels we made when we read the data files.

```
model = MultinomialNB().fit(train_matrix, train_labels)
```

Now that we've trained a model, we can try it out on a 1-star review (but we pick a review in the test set to avoid **overfitting**).

```
neg_sample_text = open(
    '{}test/neg/9999_1.txt'.format(data_path)).read()
print(neg_sample_text)
```

```
When all we have anymore is pretty much reality TV shows with people making fools of
themselves for whatever reason be it too fat or can't sing or cook worth a damn than I
know Hollywood has run out of original ideas. I can not recall a time when anything original
or intelligent came out on TV in the last 15 years. What is our obsession with watching
bums make fools of themselves? I would have thought these types of programs would have
run full circle but every year they come up with something new that is more strange
then the one before. OK so people in this one need to lose weight...most Americans need
to lose weight. I just think we all to some degree enjoy watching people humiliated.
Maybe it makes us feel better when we see someone else looking like a jerk. I don't know
but I just wish something intelligent would come out that did not insult your intelligence.
```

The overall sentiment seems fairly negative. Let's see if our model agrees by computing the class probabilities (negative first, then positive).

```
neg_sample_vec = vectorizer.transform([neg_sample_text])
model.predict_proba(neg_sample_vec)
```

```
array([[ 0.81292637,  0.18707363]])
```

That seems fairly conclusive. Now let's do the same thing for a 10-star movie review in the test set.

```
pos_sample_text = open(
    '{}test/pos/9999_10.txt'.format(data_path)).read()
print(pos_sample_text)
```

```
Although I'm not a golf fan, I attended a sneak preview of this movie and absolutely loved  it. The
historical settings, the blatant class distinctions, and seeing the good and the bad on both sides
of the dividing line held my attention throughout. The actors and their characterizations were
all mesmerizing. And I was on the edge of my seat during the golf segments, which were not only
dramatic and exciting but easy to follow. Toward the end of this movie, "Seabiscuit" came strongly
to mind, although "The Greatest Game Ever Played" is far less complex a story than that film.
In both cases, the fact that the events really happened deepened my interest.
```

We would expect this review to be fairly clearly positive, based on the text alone. What does the model say?

```
pos_sample_vec = vectorizer.transform([pos_sample_text])
model.predict_proba(pos_sample_vec)
```

```
array([[ 0.33913831,  0.66086169]])
```

The class probabilities are closer to one another than with the previous test review, but the positive sentiment is strongest, which is a good sign.

Even though it's not from a review, let's see how the model would deal with a tricky sentence with a "not" in it.

```
stuff = vectorizer.transform(
    ['A ten pound laptop is not a good travel companion.'])
model.predict_proba(stuff)
```

```
array([[ 0.5583131,  0.4416869]])
```

We don't have the greatest confidence in the prediction, but it yields the correct classification.

Performance Evaluation

But it's not enough to try out the sentiment analysis on 1 or 2 reviews: we need to know how well the model performs on the 25,000 test cases?

We'll need to load the test documents before we can compute some evaluation metrics.

```
test_docs = []
test_labels = []

pos_file_names = glob.glob('{}test/pos/*.txt'.format(data_path))
for file_name in pos_file_names:
    test_docs.append(open(file_name).read())
    test_labels.append(1)
neg_file_names = glob.glob('{}test/neg/*.txt'.format(data_path))
for file_name in neg_file_names:
    test_docs.append(open(file_name).read())
    test_labels.append(0)
```

We get the feature vectors for the test data and the model's predictions. Note that we use `transform()` on testing data rather than `fit_transform()`.

```
test_matrix = vectorizer.transform(test_docs)
predicted = model.predict(test_matrix)
```

We can evaluate the **precision**, the **recall**, and the F_1−**score** on the test set. Precision is the fraction of predicted positive results that are actually true positives, whereas recall is the proportion of true positives that are recognized as such by the classification model. Ideally, both of these values would be near 1.

The F1-score is the **harmonic** mean of these quantities.

```
print(metrics.classification_report(
    test_labels, predicted, target_names=['neg', 'pos']))
```

```
             precision    recall  f1-score   support

        neg       0.77      0.87      0.82     12500
        pos       0.85      0.74      0.79     12500

avg / total       0.81      0.81      0.80     25000
```

The performance metrics are actually quite good!

Vader

NLTK comes with a **pre-trained** sentiment analyzer called `vader`. Pre-trained in this context means that it has been trained on a dataset that does not necessarily contain positive and negative movie reviews.

We'll see how it performs on the testing set, but first we'll try it on the trick sentence from above.

```
sia = SentimentIntensityAnalyzer()
sia.polarity_scores('A ten pound laptop is not a good travel companion.')
```

```
{'compound': -0.3412, 'neg': 0.256, 'neu': 0.744, 'pos': 0.0}
```

We see that `vader` wasn't fooled: it recognizes that it's likely to be a neutral sentence, or possibly a negative sentence, but not a positive sentence.

To evaluate on the test data, we find the prediction for each test document, and load them into `classification_report` (remember that it hasn't been trained on the movie review data).

```
vader_predicted = []
for doc in test_docs:
    scores = sia.polarity_scores(doc)
    if scores['pos'] > scores['neg']:
        vader_predicted.append(1)
    else:
        vader_predicted.append(0)
```

We get the following performance metrics.

```
print(metrics.classification_report(
    test_labels, vader_predicted, target_names=['neg', 'pos']))
```

```
              precision    recall  f1-score   support

         neg       0.79      0.52      0.63     12500
         pos       0.64      0.86      0.74     12500

avg / total       0.72      0.69      0.68     25000
```

It is not surprising to see that the results are not quite as good, as the model has not been trained on movie reviews. Still, it is not the worst model in the world.

27.5 Exercises

1. How important are visual cues in communications and business negotiations? How important is context? How easy is it to learn from someone whose context is different from yours (culturally AND professionally)?
2. Conduct a BoW analysis of the Ottawa Senators' 2016-2017 NHL season (as in Section 27.4.1) using the fields `SSS_Recap` and/or `OPP_Recap`.
3. Conduct a BoW analysis of the Ottawa Senators' 2016-2017 NHL season (as in Section 27.4.1) using the fields `AP_Headline`, `SSS_Headline`, and/or `OPP_Title`.
4. Identify the plays linked to each Gutenberg Project ID in Section 27.4.2.
5. Build a Shakespeare word cloud for the Marlowe NAs, and *vice-versa*, as found in Section 27.4.2.
6. Re-run the relevant analyses of Section 27.4.2 after having cleaned the datasets of *theatre* instructions (exit, exeunt, enter, scene, act, folio, dramatis, personae, etc.) and of *copyright/licensing information* and/or *modern contaminating terms*, and with a slightly more restrictive list of early modern english stopwords, removing ancient spelling conventions ("haue" instead of "have", "goode" instead of "good", etc.).
7. Conduct a count of bigrams per play with the Shakespeare corpus of Section 27.4.2.
8. Find the most common title of nobility in each of the Shakespeare plays of Section 27.4.2.
9. Re-run the $n-$gram code of Section 27.4.2 for $n = 3$.
10. Create a classifier for real news/fake news, as in Section 27.4.4, using the `fake_or_real_news1_utf8.csv` file (found in the usual location).
11. Conduct the cluster analysis of Section 27.4.5 using stricter sparsity levels: 90%, 95%, 99%.
12. Conduct a cluster analysis on the data of Section 27.4.5 using the tf-idf weightings (without removing "Ottawa" or "Senators" from the cleaned up recap, using the following as a starting point.

```
# Apply tf-idf weighting
AP.recaps.tfidf_tdm.1 = tmLLTermDocumentMatrix(clean_corp.AP.recaps.1,
                      control=list(weighting=weightTfIdf))

# Remove sparse terms
AP.recaps.tfidf_tdm.1 <- tm::removeSparseTerms(AP.recaps.tfidf_tdm.1,sparse=.7)
```

13. Conduct the cluster analyses of Section 27.4.5 using a DTM instead of a TDM (in essence, finding similar games during the 2016-2017 season).
14. Conduct the cluster analyses of Section 27.4.5 using $k-$means instead of hierarchical clustering.
15. Run a sentiment analysis of *Macbeth* for categories of sentiments other than `positive` or `negative`.
16. Run a sentiment analysis of Trump's tweets in the BOTUS case study.
17. Run a sentiment analysis of the field `SSS_Recap` from the Senators game recap example.

Chapter References

[1] C.C. Aggarwal and C.X. Zhai. *Text Classification*. CRC Press, 2015.

[2] M.M. Alvarez. 'Can Machine Learning and AI Help Solving the Fake News Problem'. In: *The Practical Academic* 12 (2017).

[3] D.C. Anastasia, A. Tagarelli, and G. Karypis. *Document Clustering: The Next Frontier*. CRC Press, 2014.

[4] C. Arthur. 'The big bang visualisation of the top 140 Twitter influencers'. In: *The Guardian* 13 (Sept. 2010).

[5] T. Basu. *NPR's Fascinating Plan to Use A.I. on Trump's Tweets*.

[6] BBC's 60 Seconds Shakespeare. 'Teachers Themes List'.

[7] D. Bevington, ed. *The Complete Works of Shakespeare*. 5th. Longman, 2003.

[8] S. Bird, E. Klein, and E. Loper. *Natural Language Processing with Python*. O'Reilly, 2009.

[9] P. Boily, S. Davies, and J. Schellinck. *The Practice of Data Visualization*. Data Action Lab, 2023.

[10] D. Carnegie and G Taylor, eds. *The Quest for Cardenio: Shakespeare, Fletcher, Cervantes, and the Lost Play*. Oxford University Press, 2012.

[11] Twitter Counter. 'Top 100 Most Followed Twitter Accounts'.

[12] K. Dailey. 'History of Cardenio: Is Shakespeare's Lost Work Recovered?'. In: *BBC News* 12 (2012), p. 2017.

[13] N. Dieker. 'Planet Money's BOTUS Bot Has Yet to Make a Single Stock Trade'. In: 12 (Sept. 2017), p. 2017.

[14] Forbes. 'The World's Biggest Public Companies List - Forbes 2000'.

[15] A. Goldmark. *763: BOTUS*.

[16] S. Greenstone. *When Trump Tweets, This Bot Makes Money*.

[17] InvestorGuide.com. 'List of Publicly Traded Companies'.

[18] 'Company List: NASDAQ, NYSE, & AMEX Companies'. Sept. 2017.

[19] 'Cardenio Play: Overview and Resources'.

[20] N. Ide and J. Veronis. 'Introduction to the special issue on word sense disambiguation: the state of the art'. In: *Computational Linguistics* 24 (1998), pp. 2–40.

[21] M. Ingram. *Here's What a Trump Tweet Does to a Company's Share Price*.

[22] K. Jarmul. *Detecting Fake News with Scikit-Learn*.

[23] K. Jarmul. *Natural Language Processing Fundamentals in Python*.

[24] M.L. Jockers. *Text Analysis with R for Students of Literature*. Springer, 2014.

[25] D. Jurafsky and J.H. Martin. *Speech and Language Processing (2nd ed)*. Pearson, 2009.

[26] T. Kwartler. *Text Mining: Bag of Words*.

[27] D. Lee. *How Twitter Changed the World, Hashtag-by-Hastag*.

[28] A. Long et al. *The Complete Works of William Shakespeare (Abridged): Acting Edition*. Applause, 1987.

[29] G. McIntire. *Fake/Real News Dataset*.

[30] G. McIntire. *On Building a "Fake News" Classification Model*.

[31] K. Mettler. ''Trump and Dump': When POTUS tweets and stocks fall, this animal charity benefits'. In: *Washington Post* 19 (Sept. 2017).

[32] Y. Miyao. *Enju syntactic parser*.

[33] M. Murray. *Topic Modelling Financial News with Natural Language Processing*.

[34] M. Murray. *Topic Modelling Financial News Dataset*.

[35] M. Ogneva. 'How Companies Can Use Sentiment Analysis to Improve Their Business'. In: *Mashable* (2010).

[36] Christopher Potts. 'On the negativity of negation'. In: *Semantics and Linguistic Theory* 20 (2010), pp. 636–659.

[37] *Sens rally after blowing lead, beat Leafs to gain on Habs*. Associated Press. 2017.

[38] P. Shrivastava. 'Text Classification: Applications and Use Cases'. In: *Parallel Dots* (2017).

[39] J. Silge and D. Robinson. *Text Mining with R: a Tidy Approach*. O'Reilly, 2017.

[40] 'Text Mining: Classification, Clustering, and Applications'. In: Chapman and Hall/CRC (2009). Ed. by A.N. Srivastava and M. Sahami.

[41] Stanford Natural Language Processing Group. 'The Stanford Parser: a statistical parser'.

[42] T3. 'Trump & Dump Bot: Analyzes Tweets, Short Stocks'.

[43] A. Taylor, M. Marcus, and B. Santorini. 'The Penn Treebank: An Overview'. In: *Treebanks. Text, Speech and Language Technology*. Ed. by A. Abeillé. Springer, Dordrecht, 2003.

[44] Universal Dependencies. 'Universal Dependencies v2'.

[45] A. Way and N. Gough. 'Comparing Example-Based and Statistical Machine Translation'. In: *Natural Language Engineering* 11 (2005), pp. 295–309.

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

[47] Wikipedia. 'Natural Language Processing'.

[48] Wikipedia. 'Shakespeare Bibliography'.

[49] M. Zhou. 'Hamilton: A Text Analysis of the Federalist Papers'. In: *Medium* (2017).

28 Mining Data Streams

by **Kevin Cheung** and **Patrick Boily**

In pedagogical settings, we typically treat the data as "complete" in the sense that we assume that the data collection process is over – no new data will be added to our trove, and all of the past (available) data is used in the analyses. In a nutshell, data collection lies in the past, and data analysis lies in the future.

In practice, data is often collected **continuously** (or at least, in **batches**), and older data typically become **stale** and should perhaps not be weighed as heavily as more recent data when the analyses are **updated**.*

In complex analytical scenarios, however, it can be inefficient to replicate multiple analyses with sensibly the same data from one time point to the next. In this chapter, we discuss **data streams** and how to extract insights in such situations. More information is available in [7, 15].

28.1 Overview

We start by introducing the fundamental notions of the discipline.

28.1.1 Motivating Examples

Consider the owner of a gift shop at the Ottawa Macdonald-Cartier International Airport (YOW). She would like to keep track of the most popular items sold on each day – perhaps there are a few "hot items" that show up daily, and some outliers that only show up in the list sporadically. How could she approach this task?

Assuming that she uses some sort of electronic inventory system, she can export the daily transactions to a file for data analysis. As YOW is not a very busy airport, she might expect the file to contain only a small number of sales. Let's take a look at the data for one day.

```
df <- read.csv("dailysales.csv", stringsAsFactors = TRUE)
str(df)
df
df2 <- df['sku']  # Take just the sku column
ag <- aggregate(df2, by=list(item = df2$sku), FUN=length)
ag[ order(-ag$sku), ]  # Sort in descending order
```

* When does Netflix recognize that a user's taste in comedy have changed, say?

```
'data.frame': 20 obs. of  3 variables:
 $ sku   : Factor w/ 4 levels "A","B","C","D": 3 2 3 1 3 4 3 2 2 3 \ldots
 $ time  : Factor w/ 20 levels "08:30","08:35",..: 1 2 3 4 5 6 7 8 9 10 \ldots
 $ amount: num  9.99 4.49 9.99 12.99 9.99 \ldots
```

```
   sku  time amount
1    C 08:30   9.99
2    B 08:35   4.49
3    C 08:40   9.99
4    A 08:50  12.99
5    C 09:10   9.99
6    D 10:20  24.99
7    C 10:24   9.99
8    B 11:05   4.49
9    B 13:46   4.49
10   C 13:57   9.99
11   C 14:07   9.99
12   B 14:36   4.49
13   C 14:48   9.99
14   B 14:50   4.49
15   C 15:08   9.99
16   B 15:46   4.49
17   D 15:47  24.99
18   D 15:52  24.99
19   A 16:10  12.99
20   A 16:17  12.99
```

```
  item sku
3    C   8
2    B   6
1    A   3
4    D   3
```

But if instead of owning a small gift store at a medium-sized airport, she was the product manager of Amazon USA. She might want to see the top sold items since the beginning of the day every 10 minutes. But the solution for the YOW gift shop would be unlikely to work. Why? Well, in 2016,[1] Amazon USA had 500+ million items in their catalogue, and the site was conducting, on average, 200,000+ transactions per hour. This quickly becomes a **big data** problem.[2]

1: And it has only increased since.

2: See Chapter 30 for additional details.

28.1.2 Basic Notions

In the gift shop example, all the daily transactions can be processed in one go at the end of the day – this is an example of **batch processing**. In the Amazon USA example, the transactions are not available all at once for processing and results need to be updated in **time increments**; whether the volume of transactions is high or not, the need for **(near) real-time updates** is a characteristic of online processing.

Data stream mining algorithms are invariably **online algorithms**,[3] some of which might exploit **intermittent batch processing** over time windows. We will be looking at examples throughout this chapter.

3: "In computer science, an **online algorithm** is one that can process its input piece-by-piece in a serial fashion, i.e., in the order that the input is fed to the algorithm, without having the entire input available from the start. In contrast, an **offline algorithm** is given the whole problem data from the beginning and is required to output an answer which solves the problem at hand." [31]

Big data is typically characterized by the **5V framework**:

- we are dealing with large amounts of data (**volume**);
- with observations of different types which may not all be saved in relational databases (**variety**);
- created at high speed, but with accessibility and processing bottlenecks (**velocity**);
- whose quality and accuracy are harder to control (**veracity**), and
- which we hope to be able to turn into actionable, insightful results (**value**).

To react to information in **real-time**, the most critical component to tame is **velocity** [13]. High-velocity data streams arise, among others, in:

- communication channels;
- sensors;
- financial or sales transactions, and
- user-generated content.

Important applications (and fields of applications) include:

- autonomous vehicles;
- cybersecurity;
- fraud detection;
- health care;
- smart homes;
- mobile communication, etc.

Data Stream Models We can study data streams *via* formal mathematical models, such as an ordered pair (s, Δ) where s is a sequence of **data objects** and Δ is a sequence of **time intervals**.

Loosely speaking, a **data stream** is a sequence of data objects (usually a vector of numerical values) that arrive **online**, one at a time; we have **no control** over the **order** in which the data elements arrive. Data streams are potentially infinite in length. Access to the data objects is thus **sequential**; it is also **one-time**, meaning that once a data object has been processed, it is **archived** or **discarded**.[4]

Another important practical issue is the presence of **noise** in the data: thermostat readings from (cheap) sensors can fluctuate wildly even when the ambient temperature is constant, for instance. When real-time responses are expected (such as may be the case in autonomous vehicles, say), noisy data has to be carefully handled: **smoothing techniques** such as Kalman filters might be required.[5]

In practice, analysts use **data management systems** (DMS) to extract insight from data streams:

1. a **stream processor** (such as Apache Kafka) to manage input and output streams;
2. a **working storage** (limited in size) to hold recent/current data elements;
3. an **archival storage** (such as Apache Hadoop) to store past data elements;
4. a **query processor**, and
5. an **analytics API**.

4: Although a small number of data objects can be held in memory for debugging or validation purposes.

5: We will not, however be covering smoothing techniques in detail; see Chapter 9 (DUDADS, Volume 1) and [7, 15] for more information.

Note that we will not be using a DMS in this chapter's example.

Because of the velocity and volume of data in a typical data stream, exact analysis methods are often contra-indicated; instead, we use **probabilistic** or **approximation** algorithms.

Filtering Say we run a domain name (DNS) server for Internet protocol (IP) address lookup; we might need to filter millions of URLs for malicious websites. It becomes quite impractical to look up the database for every DNS query. A simple, well-known, and practical approach for this task is to employ a **Bloom filter**.

A probabilistic **memory-efficient** procedure that allows for **fast querying**, a Bloom filter is primarily used to **test whether an element** $s \in T$ **is also a member of a set** S; Bloom filters can result in **false positives** (falsely indicating that an element is in the set when it actually isn't), but they cannot produce **false negatives** – if a query indicates an element is not in the set, then it definitely is not.

Bloom filters are particularly useful in situations where the set of keys is large and expensive to enumerate; they are often used in situations where the **speed** and **efficiency** of the query process are more critical than the absolute accuracy of the result.

Bloom filters use **hash functions** $h_1, \ldots, h_k : T \to \{0, \ldots, m-1\}$ to map elements $s \in T$ to positions in an array F of m bits, where $m \gg 1$.[6] For each **known** $s \in S$, we compute the k hashes $h_1(s), \ldots, h_k(s) \in \{0, \ldots, m-1\}$; these values become the indices of F for which the bit entry is 1.

6: Hash functions are **deterministic**, **efficient** (the hash value can be computed quickly), **non-invertible** (it should be difficult to reverse-engineer the original input from the hash value), and **collision-resistant** (it should be improbable for two different inputs to produce the same output).

For instance, say $m = 12$ and there are $k = 3$ hash functions, with the set S of malicious IP containing only 4 elements.

URL	h_1	h_2	h_3
bad.com	0	6	4
malware.org	1	4	7
virus.ca	1	2	10
breach.biz	0	6	10

The set of hash values on all known malicious URLs is thus

$$\{h_1(s_1), \ldots, h_3(s_4)\} = \{0, 1, 2, 4, 6, 7, 10\}$$

and the array F (we count from index 0) is:

1	1	1	0	1	0	1	1	0	0	1	0

Consider an element $t \in T$. If any of the bits of F indexed by the hash values $h_1(t), \ldots, h_k(t)$ is 0, then $t \notin S$; otherwise, we "predict" $t \in S$ (but this could be a false positive).

For instance, say that a new URL t is received for DNS lookup; we compute $h_1(t) = 1$, $h_2(t) = 3$ and $h_3(t) = 7$:

1	1	1	0	1	0	1	1	0	0	1	0

As $F(3) = 0$, we would conclude that t is not malicious.

The size m of the Bloom filter and the number k of hash functions used affect its efficiency and the probability of obtaining false positives; a larger filter size generally results in a lower error rate, but the likelihood of false positives increases as more elements are tested. For more information, refer to [16, 22, 28].

Sampling Sampling from a data stream differs significantly from sampling from **static storage**, mostly with respect to **data accessibility** and **methods**.[7]

7: See Chapter 10 (DUDADS, Volume 1).

In static contexts,

- the data is **stable** and the dataset size is **known**, which simplifies sample size and estimator variance computations;
- the same dataset can be sampled **multiple times**, using traditional and straightforward sampling methods, and
- there is typically **no urgency** to analyze the data, as it does not change in real-time.

In streaming contexts,

- the data is **transient**, flowing in (near) real-time, and is not typically stored long-term (unless it is captured/archived during the stream);
- items often have a single chance to be sampled because they may not be seen again, necessitating **one-pass algorithms**;
- the sample may need to be **continuously updated** to reflect the most recent data;
- the total size of the data is not known in advance, which affects **sample size considerations**, and
- typically, the data must be analyzed **immediately**, since storing/archiving large amounts of streaming data may prove impractical or impossible.

Different sampling methods exist depending on the application. We illustrate two common sampling methods for data streams: **reservoir sampling** and **sampling *via* a hash function**.

Consider a credit card company, say, that wants to obtain a sample of 1000 transactions over the next hour drawn uniformly at random. But the company doesn't know how many transactions will occur during the hour, they cannot save too many transactions in memory, and accessing the archive of saved transactions may be too costly. How could they approach the task?

Assuming that the number of transactions is sufficiently larger than the **reservoir capacity** $\Lambda(= 1000)$, **reservoir sampling** can be shown mathematically to produce a uniform sample of all transactions. Start by preparing a **reservoir storage** R for $\Lambda(= 1000)$ transactions, and insert the first Λ transactions in R.

For each transaction t that follows in the specific time frame:

1. draw a random integer M in $\{1, \ldots, t\}$;
2. if $M \leq \Lambda$, randomly delete a transaction from R and replace it with the transaction t.

The transactions in R after the final transaction form the **reservoir sample**.

If instead we want to sample $r = 1\%$, say, of the credit card users and compute each sampled user's total spending over a specific time frame, we could use the following approach. Assume that we have a reasonably accurate estimate $\hat{N}$ of the total number of users who will conduct a credit card transaction during the time-frame.

As a transaction arrives, we check if the user is already in the sample:

- if they are, we simply add the transaction amount to the running total for that user;
- if they are not, then with probability r, we add the user (and the transaction value) to the sample.

The sampling procedure stops once there are more than $r\hat{N}$ users in the sample.

This method has a huge drawback, however: checking if a transaction's user matches one in the sample could **take too much time** relative to the rate at which the transactions arrive – data base lookup needs to be avoided as much as possible.

Instead, let h be a (quickly computable) **hash function** mapping credit card users to an integer in the range $\{0, \ldots, \hat{N} - 1\}$ and set up an array A of size $r\hat{N}$ to track the sampled users' total spendings.

When a transaction (u, a) arrives, we compute $h(u)$, where u is the user. If $h(u) < r\hat{N}$, we add the transaction amount a to the element of A indexed by $h(u)$. Otherwise, we leave A unchanged.

For example, say we expect $\hat{N} = 1000$ users to conduct transactions in the specified time frame. Then the array A is of size $r\hat{N} = 0.01(1000) = 10$ and only the spending amounts of users hashed to a value in the range $\{0, \ldots, 9\}$ would be tracked.

Time Windows In theory, data streams are **potentially infinite**.[8] In practice, we often use a **window** (i.e., a contiguous subset of the data) on the stream for analysis.

The most commonly time windows are:

1. **landmark** windows, which consider data from the beginning of the stream to the present;
2. **sliding** windows, which consider data from w time units before the present to the present;
3. **fading** windows, which assign smaller weights to older data, and
4. **tilted time** windows, which use different time granularities.[9]

Instead of figuring out the most popular items each hour (as in Section 28.1.1, for instance), we could monitor the situation **continuously** by assigning scores to items in such a way that the score of an item decreases over time if it is not purchased again. The popularity of an item can then be simply measured as the total score of the item, with popular items having the highest scores. We illustrate how a **fading window** could be used to accomplish the task.

Let c be a very small number, such as 0.0001 or $1/N$, where N is the number of items in the catalogue. Assume that there is a list of items

8: Data streams must necessarily be finite, but their size could be so large that it makes more sense to model them as infinite.

9: The fundamental idea is to create a hierarchy of time windows of varying sizes that reflect the different ages of the data. In more recent time periods, the windows are smaller, capturing data at a fine granularity, which allows for detailed analysis of the most recent events. As time progresses, the windows become larger, encapsulating older data at a coarser granularity. This approach allows for efficient storage and query of data streams, retaining more detail for recent data while still maintaining a summary of older data.

whose popularity we are tracking, each of which is assigned the initial score of 0. When a purchase takes place:

1. multiply each list item's score by $1 - c$;
2. if the purchased item is in the list of items, add 1 to its score;
3. for any item whose score falls below a threshold t, set the score to 0.

For instance, suppose that $c = 0.2$ and $t = 0.5$, and that the current list stands at:

Item	Eggs	Milk	Jam	Bread
Score	0.64	0.8	1	0

If the next purchase in the specified time frame is bread, then the list is updated to:

Item	Eggs	Milk	Jam	Bread
Score	0.512	0.64	0.8	1

If the next purchase in the specified time frame is milk, then the list becomes:

Item	Eggs	Milk	Jam	Bread
Score	0.4096	1.512	0.64	0.8

As the eggs' score falls below $t = 0.5$, it is set to 0.

Item	Eggs	Milk	Jam	Bread
Score	0	1.512	0.64	0.8

If this was the end of the specified period, the most popular purchased item would be milk, followed by bread, and then jam. Note that a score of 0 does not necessarily mean that the item in question has not been purchased (as was the case with eggs).

This algorithm is such that the sum of all scores prior to a purchase is at most $1/c$, and that there cannot be more than $2/c$ items in the list with a score above 0.

Learning and Validation Since the data objects in a data stream are unavailable for analysis or machine learning **all at once**, we need different approaches than those suggested in Chapters 6-11 ([8, vol. 1]) and Chapters 19-23 ([8, vol. 3]); instead, to get around this limitation, we typically use:

- **incremental learning**, in which the model is updated as new data arrive, or
- **two-phase learning**, where we periodically conduct (offline) learning on a synopsis/subset/summary of the data, and then update the model appropriately.

For similar reasons, the traditional model validation approaches (such as $k-$fold and leave-one-out validation methods) do not apply to data streams; instead, we may use **hold-out** or **prequential** validation.

In **hold-out validation**, we set aside a portion of the incoming data as a test set and use the the remainder to train the model. Since data streams are **infinite** (in theory) and **non-stationary**;[10] as such, the model may need to **update continuously**.

10: The underlying distribution of the data changes over time (see [8, vol. 1, ch. 9]).

The hold-out method is often used as a foundation for experimental frameworks in data stream analysis because it allows for **flexibility** in capturing various statistics of interest and can be adapted to track many behaviours of the model under evaluation.[11]

11: For instance, it can be set up to monitor the model's accuracy against the number of training instances processed or the actual training time, which can then be represented in **learning curves** or **performance tables**.

In practice, the hold-out method measures the **immediate accuracy** of the model at specific points in the stream (sampled according to some appropriate model) without considering the history of previous performance, which can be problematic since early poor performance can affect the accuracy measurement, even if the algorithm improves significantly over time.

But with a "**large enough**" hold-out test, we can hope to improve the reliability of the accuracy estimates; averaging the evaluation procedure multiple times over different test sets may also help in reducing the variance in the estimates.

Data mining analysts also test the performance of models on historical data. However, in real-life applications, what works well on historical data might not work well in production. When **concept/model drift** is likely to be encountered,[12] **prequential** validation (also known as the **interleaved test-then-train method**), which attempt to maintain a summary of recent samples while discarding older ones to better reflect the current distribution of the data stream, might provide a better framework for model validation.

12: That is, when the trained model might have been great for historical data, but is unlikely to be useful for new observations due to a shift in the data distribution or the way the various features interact in the data over time.

Prequential validation works as follows:

1. as each instance from the data stream arrives, it is first used to **test** the model, meaning the current model's prediction for this instance is evaluated before it is incorporated in the training set;
2. after the prediction and its evaluation, the same instance is used to further train or update the model, reflecting a real-world scenario where models must predict future or unseen data;
3. the performance of the model is tracked continuously over time, after each each prediction, using any relevant evaluation metrics.

Some variations of the prequential method involve using a **sliding window** or **fading factors** to give more weight to recent data, helping the model adapt quickly to **concept drifts** by "forgetting" older data.[13]

13: Even without these variations, the prequential approach allows the model to be constantly updated with the most recent data, which also protects against model drift.

One major advantage of using the prequential method is that it simulates a scenario where each instance can only be used once, due to the potential infinite length and high-velocity nature of streams. It **mimics the way models are used in the real world**, continuously learning and adapting from a stream of incoming data and may provide a realistic assessment of a model's predictive performance and its ability to adapt over time.

28.2 Change Detection and Maintaining Statistics

There are many phenomena/situations that exhibit **statistical** change over time:

- the demographics of a neighbourhood;
- the average room temperature in a house;
- weekly movie theatre attendance;
- the vital signs of an individual.

For each of these examples, we could expect the **summary statistics** to remain relatively stable over a "small enough" time window. However, it is clear that there are events that will cause noticeable changes in the summary statistics over a long time period.

For example, in the case of vital signs, illnesses ranging from a flu to a heart attack can trigger changes that vary from mild to wild (including the sudden disappearance of any sign).

Time Series Analysis vs. Data Streams Mining In [8, vol. 1, ch. 9], we would have tackled such situations using **time series analysis** – how does data streams mining differ? Broadly-speaking, the focus of time series analysis is on **understanding historical data** and **predicting future values** in a fixed dataset, while data streams mining is concerned with handling and extracting knowledge from an **ongoing flow** of data that may change **unpredictably** over time.[14]

14: In particular, **time series analysis** involves **ordered** sequences of **stationary** values typically measured at **successive points** in time spaced at **uniform time intervals**; the analytical aims are to understand or model the **underlying structure** and **behaviour** of the data, often to forecast future points in the series. Time series analysis uses methods that exploit **temporal correlations** (ARIMA, exponential smoothing, Fourier analysis, etc.); in general, the analysis is performed on a **complete** dataset, and it is usually **retrospective**.

In contrast, **data streams mining** deals with large volumes of (possibly **non-stationary**) data that arrive **continuously** over time, often at **high velocity**, in a potentially **never-ending process**; it aims to extract **actionable insights** from **live data** as it is generated. Data streams mining uses algorithms that process and summarize data **on the fly** (online learning, sliding windows, and approximation algorithms); in general, mining is performed in **real-time** and is often **prospective**, focusing on immediate insights and rapid decision-making.

28.2.1 Change Detection

In this section, we will take a look at **change detection** (or **concept drift detection**) over data streams. The main challenge in change detection is determining **when** a significant change occurs and modifying the data stream model appropriately to reflect the change – part of the difficulty is that **noise** may wrongly be seen as change by the untrained eye.

Change detection then needs to balance **sensitivity to change** and **robustness to noise**.

Technical Framework A data stream is typically modeled as a sequence of **data-generating distributions** from which **instances** (data objects/items) are drawn.

We denote by $\mathcal{D}_i$ the distribution for the $i-$th item in the data stream; the actual item generated from $\mathcal{D}_i$ is denoted by x_i – in other words, x_i is obtained by **sampling from** $\mathcal{D}_i$. Note that x_i could be a vector, and not necessarily just a scalar.[15]

For instance, if the distributions $\mathcal{D}_i$ are independent $\mathcal{N}(0,1)$ for $i \leq 6$, while the (remaining) distributions $\mathcal{D}_i$ are independent $\mathcal{U}(0,1)$ for $i \geq 7$, then a **realization** of the data stream could be

$$0.56, -2.43, 0.05, 1.78, -0.82, -0.21, 0.2, 0.9, 0.07, 0.54, 0.49, \ldots$$

15: In other chapters, we differentiated the scalar x from the vector $\mathbf{x}$ by using a bold font. In this chapter, we will stick to the regular font weight – if a vector is intended, the context should make it clear.

We would like to be able to detect that the initial portion ($i \leq 6$) and the tail ($i \geq 7$) of this sequence arise from **different** distributions – in other words, that a change has occurred in the mechanism to generate the data between $i = 6$ and $i = 7$.

Learning Distributions In practice, the distributions $\mathcal{D}_i$ are unknown and need to be learned from the instances x_i. When the distributions change from one instance to another in an **haphazard manner**, which is an extreme case, learning is not possible.

The other extreme case is when the distributions **remain the same** throughout the data stream. In this case, once we have "enough" data instances, we can obtain a model whose accuracy is bounded below by the theoretical limit of the **Bayes error**.[16]

16: See Chapter 21 (DUDADS, Volume 3).

In either of these extremes, there is no need for change detection algorithm (although for different reasons).

Concepts and Terminology What typically happens in a data stream is that there are stretches of the sequence in which the distributions are (nearly) identical so that a model for the underlying distribution can be learned from the examples.

We can then partition the (nearly) identical distributions into sequences of distributions.[17]

17: Of course, this may be easier said than done...

In the following sequence of distributions, let's say that the ones with the same colour are (nearly) identical:

$$\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_5, \mathcal{D}_6, \mathcal{D}_7, \mathcal{D}_8, \mathcal{D}_9, \mathcal{D}_{10}, \mathcal{D}_{11}, \mathcal{D}_{12}, \ldots$$

Denote the sequences of distributions by

$$\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \ldots$$

A model that is trained with training instances from $\mathcal{S}_1$ will perform poorly on $\mathcal{S}_2$. More generally, the distributions in $\mathcal{S}_j$ will have **different characteristics** from those of the distributions $\mathcal{S}_{j+1}$. We say that each $\mathcal{S}_j$ corresponds to a data stream **concept**; the transition from $\mathcal{S}_j$ to $\mathcal{S}_{j+1}$ is where a **concept drift** appears.

Of course, it might not be easy to distinguish a truce concept drift from the presence of **temporary noise**. In [17], the authors define concept drift in terms of **consistency** and **persistency**.

Loosely speaking, **consistency** means that the changes between consecutive instances in the same concept are small enough to stay under some pre-sepecified threshold.

Persistency, on the other hand, means that the concept is present for at least p consecutive instances where $p \geq w/2$ from some pre-specified window size w.

A concept drift is **permanent** if it is both consistent and persistent.[18]

18: Note that concepts may re-occur: in the concept sequence $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \ldots$, it is possible for the first and third concept to be identical, for instance.

Drift Detection In practice, there are two main approaches to drift detection:

- monitoring the evolution of **performance indicators**, and/or
- monitoring **distributions** on two (or more) different time-windows.

Common algorithms include: **floating rough approximations** (FLORA), **cumulative sum** (CUSUM), and the **Page-Hinkley test** (PH).

FLORA: the details of the algorithm are outside the scope of the chapter but can be found in [5, 7, 27]. Note, however, that FLORA:

- uses **"rough" set theory** (a generalization of regular or "crisp" sets from computer science);
- can adapt to **changing data patterns**, which is crucial for handling concept drift in streaming data;
- supports **incremental learning**, updating its model as new data comes in;
- is designed to identify **significant deviations** from existing models;
- is efficient and scalable, making it suitable for **big data streams**;
- is capable of handling **noise** and **uncertainty** effectively, and
- allows **user-defined parameters** to tailor detection sensitivity.

CUSUM is particularly effective in identifying shifts in the mean or variance. More details about the algorithm can be found in [5, 7, 14]. As an algorithm, CUSUM:

- is aimed at detecting **shifts in statistical properties** of a data stream (especially the mean);
- tracks the **cumulative sum of deviations** from a target or reference value;
- involves **threshold setting** for change detection – exceeding these thresholds indicates potential drift;
- implements a **two-sided approach** for detecting both positive and negative shifts;
- includes a **reset mechanism** after detecting a change, allowing for continuous monitoring;
- is **simple**, has **minimal computational requirement**, and is effective in detecting **gradual changes**, but
- can be sensitive to **parameter settings** and may require tuning.

PH focuses on identifying changes in the mean [7]:

- its primary objective is to detect **shifts in the mean** of a sequence of observations;
- it is based on a **cumulative sum approach**, but focuses on rapid detection of mean changes;
- it computes the **cumulative sum of differences** from the mean and adjusts it over time;
- it uses a **threshold** for change detection; exceeding this threshold indicates a potential mean shift;
- its implementation is straightforward and it offers **timely (rapid) detection**, but
- its effectiveness depends on the **choice of threshold** and mean estimates.

28.2.2 Maintaining Statistics

It most applications, analysts are interested in tracking data stream statistics over time. In theory, as long as we have saved all of the stream's numerical values, we can compute the mean and variance (to name but two); in practice, the **direct approach** is often fraught with issues – even something as simple as identifying the number of instances in a stream can quickly devolve into a **big data problem**.

We would much rather be able to obtain a data stream's statistics **on demand** – in the lingo of data mining, this is called **maintaining** the stream's statistics – without having to compute it from scratch at any desired moment.

If we are only interested in the **mean** and the **variance** of a numeric data stream over a **landmark window** (that is, from time 0 to now), it suffices to maintain a **triple** of numbers (N, S, Q) where:

- N is the **total number** of instances seen up to now;
- S is the **sum of all the isntances** seen up to now, and
- Q is the **sum of the squares of the instances** seen up to now.

We can use the basic formulas to compute the desired statistics: the **mean** is simply S/N, while the **variance** is $Q/N - (S/N)^2$. As a new data stream instance x comes in, we only need to update the triple as follows:

- increment N by 1;
- increment S by x, and
- increment Q by x^2,

and use the basic formulas to get the mean and variance. By keeping (N, S, Q) current with the above mechanism, we get the (current) mean and variance on demand.[19]

Unfortunately, in most data stream applications, summary statistics over landmark windows are not suitable. Instead, we might be looking for summary statistics over a **recent time-window**. A mechanism for **forgetting**, **devaluing**, or **suppressing** old data is required.

ADWIN The **adaptive sliding window** (ADWIN) is an algorithm used to compute statistics over **sliding windows** [7].

It keeps a **variable-length window** of recent values, with the property that the window has the maximal length which remains statistically consistent with the hypothesis that there has been **no change** in the average value inside the window. ADWIN can thus be used to detect **change in the average value** of the data stream (non-stationarity). The basic idea is the following one.

Let W denote the window of recent values. We search for a **splitting point** that will divide W into two sub-windows W_0 and W_1, where W_0 represents **older** instances and W_1 the recent ones. We would then consider removing W_0 from W if the average value of instances over W_0 is "substantially different" from those over W_1.[20]

Consequently, we can confidently declare a change whenever the window W **shrinks**, and the mean over the existing window W is a **reliable estimate** of the current mean in the stream.

19: If we are interested in the **progression** of the mean and variance over time, we could maintain (and store) a 4−tuple $(t, N(t), S(t), Q(t))$, where t represents the **time stamp** at which new instances come in, with $N(t)$, $S(t)$, and $Q(t)$ representing N, S, and Q at t, and the mean and the variance at t being defined by $S(t)/N(t)$ and $Q(t)/N(t) - (S(t)/N(t))^2$. We may still need to store a rather large object to plot the data, say, but the mean and variance computations are not any more demanding than in the text.

20: The search for a split point (which we will describe shortly) does not need to take place upon every new arrival; it could take place every k new arrivals, for some positive integer k.

ADWIN Algorithm ADWIN requires specifying a **confidence value** $0 < \delta < 1$. We denote the sequence of data stream values by $x_1, x_2, x_3, \ldots$, where x_j only becomes available at time t_j. For simplicity's sake, we will further assume that $0 \leq x_j \leq 1$ and that x_j is drawn from a distribution D_j with **expected value** $\mathrm{E}(x_j) = \mu_j$; in practice, of course, we know neither D_j nor its expected value.

ADWIN uses a **sliding window** $W(= W^i)$, with the most recent arrival x_i going to the head of the window. We will denote the **mean of the instances over the window** W by $\hat{\mu}_W$. The size of a window W (which is to say, the number of observations in W) is denoted by $|W|$.

1. Initialize window W;
2. for each time t_j,
   a) add x_j to the head of $W(= W^j)$;
   b) while W can be split into $W = W_0 \sqcup W_1$ as in the previous discussion, with
   $$|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| > \sqrt{\frac{\ln(4|W|/\delta)}{2m}}, \quad \text{where } m = \frac{2}{\frac{1}{|W_0|} + \frac{1}{|W_1|}},$$
   then set $W = W_1$;[21]
   c) output $\hat{\mu}_{W^j}$.

21: In other words, m is the **harmonic mean** of W_0 and W_1.

Theorem: with the threshold δ, the following hold at every step t.

- False positive rate bound: if μ_j has remained constant within W up to time t, the probability that the window is shrunk is at most δ.
- False negative rate bound: if for some partition of W into $W_0 \sqcup W_1$ we have
  $$|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| > 2\sqrt{\frac{\ln(4|W|/\delta)}{2m}},$$
  then ADWIN shrinks W to W_1 (or shorter) with probability $1 - \delta$.

Histograms Loosely speaking, a **histogram** is a visual representation of frequency data as a bar graph.[22] Histograms are often used in reporting and exploratory data analysis. Constructing a histogram involves putting the observed values into **non-overlapping bins**.

22: We have discussed such charts in Chapter 7 (DUDADS, Volume 1), Chapter 18 (DUDADS, Volume 2), and [9].

In a data stream context, **up-to-date** histograms could lead to actionable insights in **real-time**. But constructing histograms from data streams can be challenging:

- since we do not know ahead of time what values new instances could take, the histogram's (long-term) **range** is unknown;
- consequently, its **number of bins** might need to be determined dynamically (which is to say, it may change as new instances arrive);
- equal-frequency bins require **sorting**, which cannot be achieved in **linear time**, and
- up-to-date histograms need to be available **on demand**, should the user request them.

In practice, we must settle for **approximate** histograms – as long as the approximations remain "decent", we can overlook the lack of exactness as the cost of obtaining real-time "insights" **when requested**.

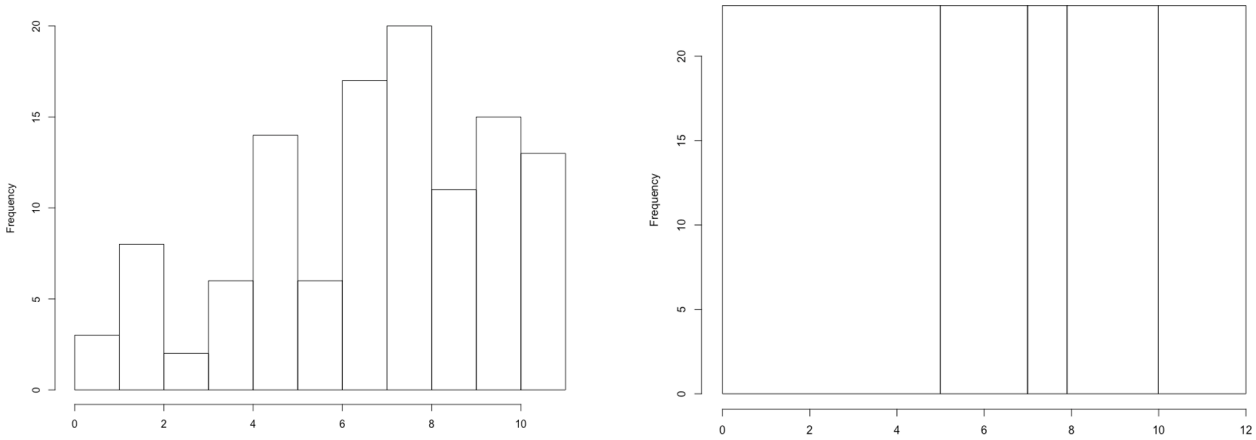


Figure 28.1: The same data is represented in two different types of histograms: **equal-width** (on the left), for which the height of the bar is proportional to the number of values in the bin, and **equal-frequency** (on the right), for which the width of the bar is proportional to the number of values in the bin.

PID The **partition incremental discretization** (PID) algorithm provides one way to maintain a data stream's histogram.

PID uses a **two-layer** (two-phase) approach:

- layer 1 obtains a **fine-grained summary** of the data objects as they arrive, while
- layer 2 **aggregates** the summary.

The idea of first obtaining a summary of the data and then performing an aggregation at lower granularity for the final result is quite common and will appear again in Sections 28.4 and 28.3.

Layer 1 We can think of layer 1 as an **intermediate histogram** summarizing the incoming data at a relatively fine granularity. It is initialized before any data is seen as follows:

1. pick a tentative but **sufficiently wide** range for the values (it can be extended later, as needed), and
2. partition the range into a **high number of equal-width intervals** (i.e., the histogram will have a large number of bins), with the first bin extending to ∞ to the left and the last bin to ∞ to the right.

Upon receiving a new instance x, we find the bin in which it falls and add 1 to that bin's **count**. If the bin count exceeds a **pre-specified threshold** (that is, if it contains more than a pre-specified percentage of the total number of values seen so far), then:

1. if the bin in question is the **first bin**, insert a bin with default width to the left of the first break point and distribute the count evenly between this new bin and the left-most bin;
2. if it is the **last bin**, insert a bin with default width to the right of the last break point and distribute the count evenly between this new bin and the right-most bin;
3. **otherwise**, split the bin into two bins with equal width and distribute the count evenly between the two bins.

The process of updating layer 1 is illustrated in Figure 28.2. As more and more instances arrive, the bins in the updated first layer are **unlikely to**

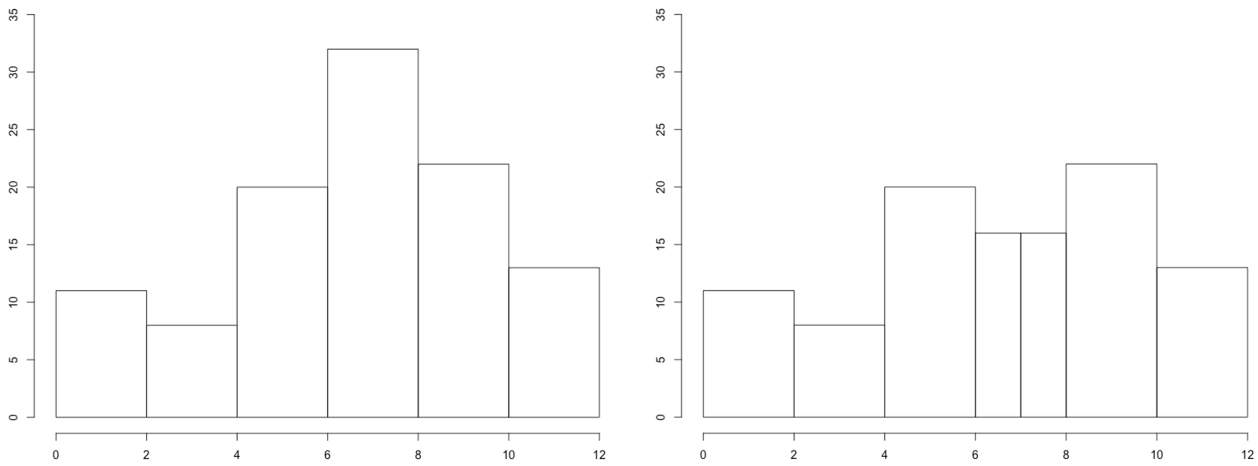


Figure 28.2: Maintaining a histogram: updating layer 1. Assume that the histogram on the left has bin $(6, 8]$ at threshold. If the next value arriving from the stream is 7.5, a bin split is triggered: the updated layer 1 is shown on the right.

be of equal width, and their number will **far exceed** the actual number of bins that would ever be displayed, both of which are acceptable as layer 1 is updated incrementally and typically stays **hidden** from the user; recall that layer 2 is only constructed from layer 1 **upon request**.

Layer 2: Equal-Width Histograms To build an **equal-width** layer 2 histogram, we select a bin width that is an **integer multiple** of the default width used for layer 1; layer 2 thus has fewer bins than the layer 1 histogram. For each bin B in the layer 2, we set the **bin count** to the tally of the bin counts in layer 1 for bins that are within the interval defined by B. For instance, consider the layer 1 histogram below.

Interval	Count
(-∞, 1]	4
(1, 2]	12
(2, 3]	8
(3, 4]	15
(4, 5]	17
(5, 6]	9
(6, 7]	10
(7, ∞)	9
(-∞, ∞)	60

Table 28.1: Layer 1 histogram for a data stream at the time of a display request from user.

Each bin in the first layer is of length 1 (apart from the first and last bin, which are of infinite length); we chose a layer 2 bin length of $3 \times 1 = 3$; the layer 2 equal-width histogram is thus as shown in Table 28.2.

Interval	Count
(-∞, 1]	4
(1, 4]	35
(4, 7)	36
(7, ∞)	9

Table 28.2: Equal-width layer 2 histogram for a data stream at the time of a display request from user.

Layer 2: Equal-Frequency Histograms To build an **equal-frequency** layer 2 histogram, we select a desired number of bins k. Let $C = N/k$, where N is the **total number of instances** captured by layer 1.

The bins in the second layer 2 are created as follows:

1. tally the bin counts in Layer 1 starting from the left-most bin;
2. as soon as the tally meets or exceeds C, insert a new breakpoint at the **right breakpoint** of the last bin of layer 1 used in the tally;
3. resume tallying bin counts starting at the **next bin** in layer 1;
4. repeat this process until all the bins in Layer 1 have been processed.

With the layer 1 histogram of Table 28.1 and $k = 3$, we have

$$C = N/k = 60/3 = 20;$$

the equal-frequency layer 2 histogram is thus as shown in Table 28.3.

Interval	Count
(-∞, 1]	4
(1, 2]	12
(2, 3]	8
(3, 4]	15
(4, 5]	17
(5, 6]	9
(6, 7]	10
(7, ∞)	9
(-∞, ∞)	60

Interval	Count
(-∞, 3]	30
(3, 5]	32
(5, ∞)	28

Table 28.3: Equal-frequency layer 2 histogram for a data stream at the time of a display request from user.

PID Revisited The bad news is that PID does not generate **exact histograms** in general because of the **granularity** of the first layer.

In particular, there could be issues with the **set of boundaries** (layer 2 breakpoints are restricted to the set of breakpoints available in layer 1) and the **frequency counts** (in layer 1, the bin counts are exact **only if no splitting takes place**).

The good news is that both layers can be constructed **efficiently**, however. When mining data streams then, as in life in general, you win some and you lose some.

PID and Change Detection One way to detect change *via* data stream histograms is to maintain a layer 2 histogram over a **reference time window** and compare with a layer 2 histogram over a **recent time window**, as distributions.

Provided that the number of bins and the widths of the two histograms are the same, the **Kolmogorov-Smirnov test** [29] can be used. Alternatively, one can also compute the **Kullback-Leibler divergence** [30] which measures the "distance" between two probability distributions.

If the histograms are deemed to be **statistically different**, then concept drift has occurred.

28.3 Clustering

Clustering is the process of partitioning data points into different groups so that data points within each group have **high similarity** whereas those from different groups have **low similarity**.

We have discussed these notions and presented various algorithms to cluster offline data in Chapters 19 and 22 of [8, vol. 3] (see also [2]).

28.3.1 Basics and Challenges

But when data is arriving **continuously** (or in small chunks), it might be useful to incorporate the new data into the existing models without re-computing everything from scratch (especially when the underlying data sets are large).

This is often conducted in **batches**: the data is collected until some threshold quantity is met, and each batch of new data is then **combined with the existing model** to create a new model.

But there are technical challenges, as we have seen: if the data arrive **continuously** in real-time (and at high velocity), then there could be time constraints. More worryingly, clusters could evolve over time, which is to say that new clusters might **emerge** or old clusters might **merge** or disappear altogether.

28.3.2 Approaches

There are two main approaches for clustering over data streams.

Partitioning In this approach, we start with k clusters obtained from an initial batch of data from the stream and we continuously update the clusters as new data points arrive, thus maintaining k clusters throughout.

In the $k-$means context, for instance, after batch t have been processed and clustered, let:

- $c_i(t)$ be the ith cluster centroid;
- $n_i(t)$ be the number of points assigned to the ith cluster;
- $x_i(t)$ be the centroid of the points batch $t+1$ closer to $c_i(t)$ than to any other centroid $c_j(t)$, and
- $m_i(t)$ be the number of points in batch $t+1$ closer to $c_i(t)$ than to any other centroid $c_j(t)$.

The **streaming** $k-$**means** algorithm updates $c_i(t)$ and $n_i(t)$ as follows:

$$c_i(t+1) = \frac{\alpha c_i(t) n_i(t) + x_i(t) m_i(t)}{\alpha n_i(t) + m_i(t)}$$
$$n_i(t+1) = n_i(t) + m_i(t),$$

where $\alpha \in (0, 1]$ is a **decay factor** used to weigh older data less relative to new data.[23]

23: Some other popular algorithms allow the number of clusters to deviate from k within a small multiplicative factor to obtain better clustering performance.

Two-Phase In this approach, we maintain many small clusters (sometimes called **micro-clusters**) that summarize the incoming data, and perform **offline clustering** on these small clusters, treating them as **weighted virtual data points**.

We will consider two methods in particular: CluStream and DenStream.

Technical Assumptions Throughout, we assume that:

- data points belong to $\mathbb{R}^d$ for some positive integer d;[24]
- the Euclidean distance is used to measure the dissimilarity between points,[25] and
- we describe clusters using a **representative**, the **number of points** in the cluster, its **density** and/or **shape**, and its **distances** to other clusters.[26]

Distance-based methods usually result in **spherical** (blob-like) clusters; density-based methods can result in **non-convex** clusters.[27] The application at hand dictates which type of cluster is most desirable.

In a data stream setting, we are unlikely to keep all the data in memory; we work instead with **data summaries**. One key idea found in many clustering algorithms is the concept of **cluster features** (CF). The simplest CF structure in use is a triple $(N, \mathbf{S}, \mathbf{Q})$, one per cluster, where:

- N is the number of points assigned to a cluster;
- $\mathbf{S} \in \mathbb{R}^d$ is vector of the sum of the variables for the N observations in the cluster, and
- $\mathbf{Q} \in \mathbb{R}^d$ is the vector of the sum of the squares of the variables for the N observations in the cluster.

For instance, suppose that the $N = 3$ points $(1, 2)$, $(2, -1)$, $(0, 1)$ are assigned to the same cluster. Then

$$\mathbf{S} = (1 + 2 + 0, 2 + (-1) + 1) = (3, 2)$$
$$\mathbf{Q} = (1^2 + 2^2 + 0^2, 2^2 + (-1)^2 + 1^2) = (5, 6),$$

so the cluster's CF is $(3, (3, 2), (5, 6))$, and that is the cluster summary that would be stored, the actual points being discarded or archived.

However the CF is defined, it should have two properties: **incrementality** and **additivity**. If a new point x is assigned to an existing cluster with CF $(N, \mathbf{S}, \mathbf{Q})$, then the new CF for the cluster is simply $(N + 1, \mathbf{S} + x, \mathbf{Q} + x^2)$ where x^2 is obtained from x by squaring each component.[28] If we merge clusters A and B, with corresponding CF $(N_A, \mathbf{S}_A, \mathbf{Q}_A)$ and $(N_B, \mathbf{S}_B, \mathbf{Q}_B)$, then $(N_A + N_B, \mathbf{S}_A + \mathbf{S}_B, \mathbf{Q}_A + \mathbf{Q}_B)$ is the CF of the merged cluster.

Being able to update CF quickly when new points are added to a cluster or when clusters merge is **critical** for being able to cluster high-velocity data streams. The importance of additivity will become apparent when we discuss CluStream.[29]

The simple CF defined above provides sufficient information to compute some **cluster metrics** that can help us decide when to merge or create new clusters.

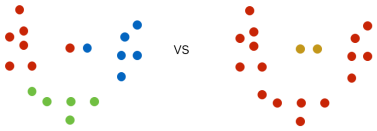
- The **centroid** (mean) of a cluster is $\mathbf{S}/N$.

24: Thus, we must embed nonnumerical data (such as text) before we can cluster the data – see Chapter 32 for examples.

25: Other options are available, of course.

26: We typically use cluster-level summary statistics to do so.

27: Convex clusters (left) vs. non-convex clusters (right), below.



28: Remember that x is a vector observation.

29: In a nutshell, the property allows us to "subtract" older micro-clusters from recent micro-clusters when clustering over a time window.

- The **radius** of a cluster is

$$\sqrt{\frac{s(\mathbf{Q})}{N} - \left\|\frac{\mathbf{S}}{N}\right\|^2},$$

  where $s(\mathbf{X}) = X_1 + \cdots + X_d$.
- The **diameter** of a cluster is

$$\sqrt{\frac{s(\mathbf{Q})N - (s(\mathbf{S}))^2}{\binom{N}{2}}}.$$

Note that these differ from the usual geometrical definitions of the radius (the average distance of the points to the centroid) and the diameter (the maximum distance between any two points in the set).

28.3.3 Evaluation

Evaluating the **quality of a clustering outcome** is of course important – we would definitely like to know how "good" the clusters returned by a clustering method are. Unfortunately, it can be difficult to come up with evaluation measures in unsupervised learning settings.

Many evaluation measures have been proposed, including **internal measures** and **external measures**.[30]

30: See discussion in Section 22.3 (see [8, vol. 3]) for more details.

Internal Measures

- **Cohesion:** the average distance from a point to the centroid of the cluster to which it is assigned (the smaller the better).
- **WSS**: the sum of squared distances from data points to their assigned centroids (the smaller the better).
- **Separation:** the average distance from a point to the points assigned to other clusters (the larger the better).

External Measures External measures require knowledge of the **ground truth**, which is to say, the "true" clustering of the data points. In practice, the ground truth is usually unavailable (assuming that there is even one in the first place); in research settings, known clusters are generated to test clustering methods.

- **Accuracy:** the fraction of the points assigned to their correct cluster.
- **Recall:** the fraction of the points of a cluster that are actually assigned to it.
- **Precision:** the fraction of the points assigned to a cluster that truly belong to it.
- **Purity:** $(f_1 + f_2 + \cdots + f_c)/N$ where f_i is the maximum number of points in the computed cluster i belonging to the same true cluster; the highest possible value is 1.[31]

31: A drawback of this measure is that it can be gamed by assigning each point to its own cluster.

28.3.4 Algorithms

Current popular data stream clustering methods follow a framework similar to the one proposed in [3]. The framework consists of **two phases**:

1. we first create and maintain a **data abstraction** (summary structure, sketch, microclusters, etc.) to hold a summary of the incoming data at a reasonably **fine granularity**;[32]
2. next, we invoke a traditional clustering algorithm on the data abstraction to obtain a small number of **global macro-clusters**, upon request by the user.

32: For example, there can be millions of micro-clusters even though we may want no more than 10 clusters in the end.

CluStream The original algorithm of [3] maintains multiple **micro-clusters** and it clusters over an **approximate time window**.[33]

33: We describe an updated version of CluStream which uses the **geometric time frame**, which is simpler to implement than the original **pyramidal time frame**. The former is described in [4].

CluStream requires an extended CF which includes **temporal summary statistics**; for each micro-cluster, T is the sum of the time stamps of the points assigned to the micro-cluster and T_Q is the sum of the squares of those same time stamps.

At any given time, CluStream maintains q micro-clusters, where q could be quite large, as long as we can store the micro-clusters in memory. The q micro-clusters are **initialized** by applying an offline clustering method (such as $k-$means, say) to the first batch in the data stream, and they are assigned a unique ID.

After initialization, whenever a new data point arrives, it is either

- **absorbed** into an existing micro-cluster, or
- a micro-cluster is **created** solely for this point.

Which of these options is selected depends on the distance of the new point to the existing micro-clusters.

When a new micro-cluster is called for, we need to either **delete an old cluster** or **merge two neighbouring micro-clusters** into a single one so that we do not exceed the total number of maintained micro-clusters.[34] The decision to delete an old cluster is based on the temporal summary statistics in the extended CF of the micro-clusters.

34: A merged micro-cluster contains the IDs of the micro-clusters that were merged – these IDs are needed when clustering over a time window.

Every once in a while, a **snapshot** of the micro-clusters is taken and is stored away. The snapshot schedule follows a **geometric time frame**, leading to snapshots stored at different levels of granularity depending on how recent the data is.

If one unit of clock time (say 1 second) is the finest granularity, then each snapshot is classified with a **frame number**, a value from 0 to $\log_2(T)$, where T is the maximum length of the stream (such as $T = 20$ yrs $\approx 6.3 \times 10^8$ sec).

The snapshots with frame number i are then stored at clock times divisible by 2^i but not by 2^{i+1}. For instance, frame number 0 would contain snapshots at odd clock times, frame number 1 would contain even clock times not divisible by 4, frame number 2 would contain snapshots at clock times divisible by 4 but not by 8, frame number 3 would contain snapshots at clock times divisible by 8 but not by 16, and so on.

Let M be a relatively small positive integer. We retain/store only up to the M most recent snapshots for each frame number that will be stored; the older ones are deleted. For instance, suppose that $M = 3$ and the current clock time is 70. The table below shows the clock times of snapshots for various frame numbers.

Frame number	Clock times of snapshots
0	69, 67, 65
1	70, 66, 62
2	68, 60, 52
3	56, 40, 24
4	48, 16
5	64, 32

This approach maintains a large number of micro-clusters on-line, takes **intermittent snapshots** of the micro-clusters and stores them away according to a **geometric time frame**, and uses an offline clustering algorithm on the micro-clusters to obtain macro clusters when requested by the user.

An important property of the geometric time window is that for any time window, at least one stored snapshot can be found within a factor of 2 of the specified horizon.

Lemma: let $[h, t_c]$ be a user-specified time window where t_c is the current time. If $M \geq 2$, then a snapshot exists at a time t_s for which $h/2 \leq t_c - t_s \leq 2h$.

To cluster over the time window $[h, t_c]$ where t_c is the current time, we find a snapshot at a time t_s for which $h/2 \leq t_c - t_s \leq 2h$ (whose existence is guaranteed by the lemma) and consider the micro-clusters at t_c.

For each micro-cluster A at t_c, we identify all the micro-cluster ID associated with this micro-cluster and subtract away the CF summary stats of the corresponding micro-clusters in the older snapshot from the CF of A.

For instance, let's assume that micro-cluster A contains the micro-cluster ID 2 and 9 at t_c.[35] Next, we look up the micro-clusters with ID 2 and 9: their respective CF are $(N_2, \mathbf{S}_2, \mathbf{Q}_2)$ and $(N_9, \mathbf{S}_9, \mathbf{Q}_9)$.[36] We adjust the CF of A, $(N_A, \mathbf{S}_A, \mathbf{Q}_A)$, to $(N_A - N_2 - N_9, \mathbf{S}_A - \mathbf{S}_2 - \mathbf{S}_9, \mathbf{Q}_A - \mathbf{Q}_2 - \mathbf{Q}_9)$.

Finally, we perform **(weighted)** $k-$**means** on the adjusted micro-clusters, treating their CF as **virtual** data points: these **macro-clusters** would be the ones that are reported on demand.

The result will be a clustering over a time window that is not too far off from the user-specified time-window $[h, t_c]$.[37]

35: That is, A was formed by merging micro-clusters with IDs 2 and 9 at some point in the past.

36: We are ignoring the temporal summary stats for the current illustration but they would be tackled in the same manner.

37: Obviously, t_c could be replaced by an earlier time for which we have a snapshot. So we are not always restricted to a most current time window. This may prove useful for change detection.

DenStream We will not get into the details of DenStream, other than to say that this algorithm was proposed in [10] (see also [7]); the authors claim that it can overcome CluStream's limitation in handling **noise**, **outliers**, and **non-spherical clusters**.

DenStream uses a **fading function** to calculate micro-cluster **weights**; points that are more distant in time contribute less to the weight. Unlike CluStream, DenStream does not allow users to obtain clusters over a particular time window.

There are many other clustering algorithms over data streams – we have covered only some basic ideas and a selection of methods. This is a field of active research, with different methods addressing the demands of different applications.

Unfortunately, many algorithms published in research papers still do not have readily accessible implementations for use in production.[38]

38: This is also the case for classification algorithms.

28.4 Classification

In machine learning, **classification** is the task of assigning a category or label to a new data object.[39]

39: For instance:

- Is an image that of a cat?
- Is an e-mail message spam?
- Is a credit card transaction fraudulent?
- Is a student at risk of failing?
- Is an HTTP request a network attack?

We have discussed these notions and presented various algorithms to classify offline data in Chapters 19 and 21 of [8, vol. 3] (see also [1]).

28.4.1 Basics and Challenges

Classifiers are trained on a pre-defined set of examples. In the case of spam filtering, e-mail messages that are known to be spam and those known not to be spam (a.k.a. ham) are used for training. They key is that the categories (true labels) of the training data are known. Classifier training, unlike clustering, is a case of **supervised learning**.

Challenges The need for the true labels for the training data seems to lead to a dilemma for building classifiers over data streams. The point of having a classifier is to **predict the label** of a **new data object**. A trained classifier can predict over static data or from a data stream; in other words, once we have a trained classifier, it can be put into operation whether or not the data objects come from streams.

For data coming from a data stream, where do the true labels come from? They cannot come immediately with the data or there would be no need to build classifiers. If the true labels come after the fact, how do classifiers **adapt** to concept drift? For example, spam filters cannot be static as new types of spam messages can arise and old ones can go out of circulation.

Settings In a batch setting, examples with their true labels are stored in a data base, and **random access** to the data is assumed. A portion of these examples is set aside to form the **test set**; once a classifier is trained, it can be put into operation for prediction over a data stream or a data base.

But in a data stream setting, not all the examples may be available at once. True labels might not be available right away and may only be available some time after the rest of the data arrive.

For example, say an e-mail message arrives and the existing spam filter misclassifies it. The user marks it with the correct label (e.g., by clicking the 'Is Spam' or 'Not Spam' button) and the classifier can then be updated with this new example or along with other accumulated examples, assuming that an **incremental training method** is used.

Scenarios But we don't necessarily need methods to train classifiers over data stream only because we need to build classifiers **on the fly** – it could happen that the training examples are only accessible in a **stream-like fashion** (e.g., in a MapReduce framework) in which case multiple access to the data is very costly or infeasible.

Another reason for having training methods that operate over data streams is to deal with **concept drift**: for instance, e-mail spam can **evolve over time**, as can network attacks, fraudulent transaction patterns, etc.

We thus need training methods that can adapt from new data – we do not want to retrain from scratch every time we think the current model is out-of-date.

Obtaining True Labels In general, obtaining **true labels** is a challenging task. In the past, labeling was done manually. For example, image labels were entered by human beings looking at the image and typing in the correct label. Nowadays, crowdsourcing is one method employed by companies that offer image-sharing platforms; they scrap information from tags or captions provided by the platform users. **Clustering algorithms** are sometimes used to speed up the manual labeling process in what is called **semi-supervised learning**.

There is on-going research on how to obtain proper labels with few examples: promising avenues include **generalized adversarial networks** (GANs) and **one-shot learning** (see the work of [25]).

Evaluation It is important to be able to compare the performance of different classifiers and training methods. However, classifier training often involves **massive** amounts of data and does not come cheap. Hence, we search for a "good" **return on investment:** good performance should not come at too steep a price.

In practice, this eliminates the traditional **cross-validation** method as being too costly, computationally, in favour of other approaches, such as: **holdout**; **interleaved test-then-train**; **prequential**, or **interleaved chunks**.

28.4.2 Approaches

There are tons of classification algorithms for batch settings [8, ch.21], but deriving versions of these that can be trained from data streams is not a trivial endeavour.

In the rest of this section, we provide a brief description of two algorithms: **Hoeffding trees** and **on-demand** k**NN** (an extension of CluStream). We conclude with a few remarks on **ensemble classifiers**.

Hoeffding Trees In a classification tree, at each **intermediate** node (i.e., not a leaf), we query a specific data feature to determine which **child** to descend/**branch out** to. Once a leaf is reached, the label associated with the leaf is the label assigned to the data object under consideration.

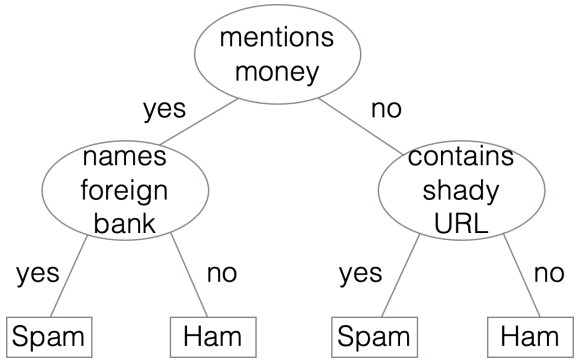


Figure 28.3: Example of a spam-filtering decision tree.

Suppose that we want to build a decision tree classifier from a set of examples with a fixed set of features. Using Shannon's information theory, we can compute the **entropy** of this set (see [8, sec 19.4.3] for details).[40]

The process of building a decision tree for classification involves selecting a feature and a condition for splitting the set into two smaller sets so that the total entropy of the two subsets is less than the entropy of the set taken as a whole. Usually, at each stage, we choose the feature that leads to the **maximum reduction of entropy**.

The process typically stops way before we reach a stage where no further splits can reduce the entropy – there are heuristics to determine when the time is ripe to **stop creating new decision nodes**.

Decision trees are very easy to build once the entire training is available and the results are **interpretable**, but once a model is found on the training data, it does not usually generalize very well to new data.[41]

In a data stream setting, we do not have access to the entire training set at any given time, however. The idea behind a **Hoeffding** tree is not to train on the entire training set (coming through a stream) but to first accumulate a "sufficient" number of observations before making decisions about splitting.

The algorithm is shown in Figure 28.4; the precise details are fairly technical and can be found in [12]. The method depends on a well-known result in probability theory called the **Hoeffding bound**, which provides a statistical guarantee that a decision made from a small data sample matches the decision that would be made if all the data was available.[42]

40: In a nutshell, the entropy of a set can be thought of as the minimum number of bits required to represent the **disorder** in the data.

41: The addition of a few new training instances could dramatically change the topology of the model Extensions (such as **boosting** or **random forests**) can be used to attempt to mitigate this shortcoming.

42: Similarly, a few flips of a coin (fewer than a hundred, certainly) are usually sufficient to give an idea of the true odds of flipping 'Heads' for that coin.

Inputs:	S	is a sequence of examples,
	$\mathbf{X}$	is a set of discrete attributes,
	$G(.)$	is a split evaluation function,
	δ	is one minus the desired probability of choosing the correct attribute at any given node.
Output:	HT	is a decision tree.

Procedure HoeffdingTree $(S, \mathbf{X}, G, \delta)$
Let HT be a tree with a single leaf l_1 (the root).
Let $\mathbf{X_1} = \mathbf{X} \cup \{X_\emptyset\}$.
Let $\overline{G}_1(X_\emptyset)$ be the $\overline{G}$ obtained by predicting the most frequent class in S.
For each class y_k
  For each value x_{ij} of each attribute $X_i \in \mathbf{X}$
    Let $n_{ijk}(l_1) = 0$.
For each example $(\mathbf{x}, y_k)$ in S
  Sort $(\mathbf{x}, y)$ into a leaf l using HT.
  For each x_{ij} in $\mathbf{x}$ such that $X_i \in \mathbf{X}_l$
    Increment $n_{ijk}(l)$.
  Label l with the majority class among the examples seen so far at l.
  If the examples seen so far at l are not all of the same class, then
    Compute $\overline{G}_l(X_i)$ for each attribute $X_i \in \mathbf{X}_l - \{X_\emptyset\}$ using the counts $n_{ijk}(l)$.
    Let X_a be the attribute with highest $\overline{G_l}$.
    Let X_b be the attribute with second-highest $\overline{G}_l$.
    Compute ϵ using Equation 1.
    If $\overline{G}_l(X_a) - \overline{G}_l(X_b) > \epsilon$ and $X_a \neq X_\emptyset$, then
      Replace l by an internal node that splits on X_a.
      For each branch of the split
        Add a new leaf l_m, and let $\mathbf{X_m} = \mathbf{X} - \{X_a\}$.
        Let $\overline{G}_m(X_\emptyset)$ be the $\overline{G}$ obtained by predicting the most frequent class at l_m.
        For each class y_k and each value x_{ij} of each attribute $X_i \in \mathbf{X_m} - \{X_\emptyset\}$
          Let $n_{ijk}(l_m) = 0$.
Return HT.

Figure 28.4: The Hoeffding tree algorithm.

If $\delta > 0$ is the decision procedure's **confidence level** and $n \geq 1$ is the **number of training observations**, then the error margin ε is

$$\varepsilon = \sqrt{\frac{\ln(1/\delta)}{2n}};$$

if $r \in [0, 1]$ is the true probability that an observation's class is A, say, and if the model built on n training observations suggests that the probability of the observations is $\overline{r} \in [0, 1]$, then

$$P(|\overline{r} - r| < \varepsilon) = 1 - \delta.$$

Practically speaking, a Hoeffding tree can confidently (at a pre-determined level δ) split an intermediate node **without having access to all the data**; as more data comes in, it may refine its decisions but it will **not reverse to an earlier state** unless there arises a "major" reason to do so.

Several versions of **Hoeffding Adaptive Trees** (which extend Hoeffding trees) are described in [6]: HAT-INC, HAT-EWMA, HAT-ADWIN; more recent work investigates the usage of drift detectors in HATs [23].

On-Demand kNN In [4], the authors proposed a framework for **on-demand classification** of evolving data streams using a **two-phase approach** as in CluStream:

- phase 1 involves the on-line maintenance of **micro-clusters**, while
- phase 2 involves an efficient **batch classifier** (kNN with small k).

kNN is simple; the user is only required to select a **measure of distance** between data points and a **positive integer** k. The training examples are stored; when a new point needs to be classified, we look for the k nearest stored examples and find their most frequent label – this becomes the predicted class for the new observation (see Figure 28.5).

But as we have seen, not all training examples are available at once in a stream setting. Additionally, there may be too many points to store in memory. Instead, we build a **summary** of the training examples in terms of **labeled micro-clusters**.[43]

When new micro-clusters are created, we may have to **delete** an old cluster or **merge** two nearby micro-clusters with the same label so the maximum number of **maintained micro-clusters** is not exceeded.[44]

As in CluStream, ID are needed when clustering over a time window. The decision to delete an old cluster is based on the temporal summary statistics in the **extended cluster features** (CF) of the micro-clusters.

To classify a new point using a classifier built with examples in the time window $[h, t_c]$, we find a snapshot at time t_s such that $h/2 \leq t_c - t_s \leq 2h$ (guaranteed by the lemma on page 1843) and the micro-clusters at the **current time**.

For each micro-cluster A at time t_c, we identify all the micro-cluster ID associated with A and subtract away the CF summary stats of the corresponding micro-clusters in the **older** snapshot provided by the CF of A.

The new point is then assigned the label of the **nearest micro-cluster**.

43: As were defined/used in CluStream, in the previous section.

44: A merged micro-cluster contains the ID of the micro-clusters that are merged.

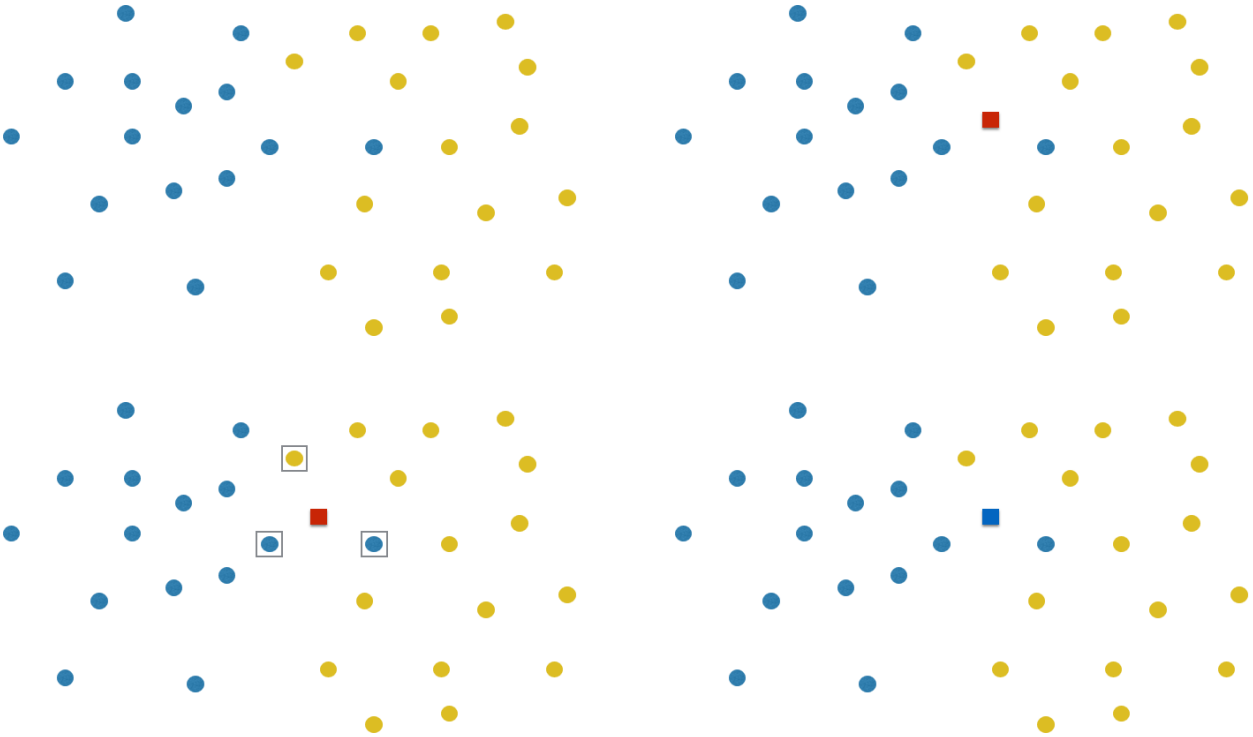


Figure 28.5: A review of kNN classification: a two-class dataset (top left); a new observation (in red) which must be classified (top right); comparing with $k = 3$ nearest neighbours (bottom left); classifying the observations as a blue square (bottom right).

28.4.3 Ensemble Classifiers

As the name suggests, an **ensemble classifier** consists of a set of individual classifiers. They do not all have to be of the same type: those that are, however, are trained on **slightly different data** and/or with **slightly different parameters**. A **(weighted) majority** of the individual classifiers provides the **ensemble classification**.[45]

45: See [8, sec. 21.5] for more details.

As an example, suppose we have an ensemble classifier for email spam consisting of three tree-based classifiers. When a new e-mail message is received, each of these three classifiers independently predicts whether the e-mail message is 'spam' or 'ham'; if **two classifiers or more** predict that the message is 'spam', then the ensemble prediction is that the e-mail message is 'spam', **otherwise** the ensemble prediction is 'ham'.

Adapting to Change The main advantage of maintaining an ensemble of classifiers in the streaming setting is that it provides the ability to handle **concept drift**.

If the performance of a classifier in the ensemble declines over time, say, then we can **decrease** its weight in the ensemble decision; if its performance becomes completely unacceptable, then it can be **removed** from the ensemble outright.

Similarly, we can add new classifiers to the ensemble to improve its overall performance without having to discard any individual classifier with (still) acceptable performance.

An early attempt to address concept drift using ensemble classifiers is provided in [24]; a more general framework is proposed in [26].

28.5 Frequent Itemset Mining

Say we have a data source of **transactions**, each of which consists of a set of items – we may be interested in discovering the common collections of items that occur among the transactions. Such item collections are called **frequent itemsets**.

Frequent itemset mining powers **association rule mining** [8, sec. 19.3]), which used to be common component of **recommender systems**.[46] An **association rule** takes the form $X \rightarrow Y$ where X and Y are sets. One possible interpretation of $X \rightarrow Y$ is "If we see X, we are likely to see Y as well."

46: In recent years, **collaborative filtering** has become more popular than association rule mining for recommenders (see Chapter 34).

It would be a mistake to treat such rules as **causal relationships**, however; data mining algorithms are typically incapable of learning **causality** (see Chapter 36 for more on this topic). At best, association rules reflect **correlations** found in data.

Example Suppose that a small supermarket sells only 4 types of products: apples, bread, cheese, and watermelons. There are $2^4 - 1 = 15$ possible itemsets of purchased goods, as can be seen in the table below.

Itemset size	Possible itemsets
1	{🍎}, {🍞}, {🧀}, {🍉}
2	{🍎,🍞}, {🍎,🧀}, {🍎,🍉}, {🍞,🧀}, {🍞,🍉},{🧀,🍉}
3	{🍎,🍞,🧀},{🍎,🍞,🍉},{🍎,🧀,🍉},{🍞,🧀,🍉}
4	{🍎,🍞,🧀,🍉}

Table 28.4: Possible itemsets in the supermarket example, when only 4 items can be purchased.

In practical situations, the number of possible itemsets is **much larger** than the total number of items – a typical supermarket easily carries 1000+ different items, so there are already at least $2^{1000} - 1 \approx 10^{301}$ **non-empty itemsets**!

In general, the number of possible itemsets is **exponential** in the number of items, and it becomes highly impractical (if not impossible) to keep track of that many itemsets. But ways have been devised to mitigate the combinatorial explosion of itemsets (see [8, sec. 9.3] and [19]); as we cannot have our cake and eat it too, some of these methods sacrifice **exactness** and can lead to **false positives/negatives**.[47]

47: This trade-off between resource consumption and accuracy of results is par for the course in machine learning and data science.

One concrete way to tame the combinatorial explosion requires drastically cutting down the number of "interesting" itemsets. A **frequency threshold** $\sigma \in (0, 1)$ is first specified; an itemset X is considered **frequent** if it occurs in at least σ (viewed as a percentage) of the available transactions. For instance, when $\sigma = 20\%$, an itemset X is considered frequent if it occurred in at least 20% of all transactions.

The **support** of an itemset is the number of transactions that contain it; if an itemset appears in two transactions, say, then its support is 2.

We say that a frequent itemset is **closed** if it has no frequent **superset** with the same support.[48] A frequent itemset is **maximal** if it has no frequent superset.

Theorem: any maximal frequent itemset is also closed, but closed frequent itemsets need not be maximal.[49]

Example Consider the same small supermarket as before, after it introduced a fifth product: cookies. Five customers have made purchases today: their transactions are summarized in the table below.

Transaction	Items in transaction
t1	{🍎, 🍞, 🧀, 🍉}
t2	{🍞, 🍉, 🍪}
t3	{🍎, 🍞, 🍉, 🍪}
t4	{🍎, 🍞, 🧀, 🍉, 🍪}
t5	{🍞, 🧀, 🍉}

Table 28.5: Transactions in the supermarket example.

Suppose that the frequency threshold is $\sigma = 0.4$ and consider the itemset $X = \{\text{bread}, \text{cheese}\}$. The support of X is 3 since it is a subset of transactions t_1, t_4, and t_5, but not of transactions t_2 and t_3; X is a frequent itemset since its support is greater than $0.4 \times 5 = 2$.

The supersets of X with three items are

$$\begin{aligned} X_a &= \{\text{bread}, \text{cheese}, \text{apple}\}, \\ X_w &= \{\text{bread}, \text{cheese}, \text{watermelon}\}, \\ X_c &= \{\text{bread}, \text{cheese}, \text{cookie}\}. \end{aligned}$$

As X_a appears only in t_1, t_4, its support is 2; as X_2 appears only in t_1, t_4, t_5, its support is 3; as X_c appears only in t_4, its support is 1. That means that the X_a and X_w are frequent itemsets for this data, but that X_c isn't; in particular, X is neither closed nor maximal.[50] □

Apriori Algorithm Before we look at frequent itemset mining in data streams, we briefly discuss the **apriori algorithm** in a batch setting (see [8, sec. 19.3] for more details). This algorithm exploits the **apriori property**, which states that all subsets of a frequent itemset are themselves also frequent.[51]

48: If $A \subseteq B$, then A is a subset of B and B is a superset of A.

49: **Proof:** for the first statement, let X be a maximal frequent itemset that is not closed. Then there is a superset $Y \supseteq X$ with the same support as X. Thus, Y must be a frequent itemset, contradicting that X has no frequent superset. For the second statement, suppose that the support level for being frequent is 3. It could be that there is some closed frequent itemset X with support 4, but every frequent superset of X has support 3. ■

50: Note however that

{bread, watermelon, cookie}

is maximal.

51: The converse of that statement is that all supersets of an infrequent itemset are themselves also infrequent.

Suppose that $X = \{\text{bread}, \text{cheese}, \text{apple}\}$ is a frequent itemset in a transaction dataset; according to the apriori property, the following itemsets are also frequent:

$$\{\text{bread}, \text{cheese}\}, \{\text{bread}, \text{apple}\}, \{\text{cheese}, \text{apple}\},$$
$$\{\text{bread}\}, \{\text{cheese}\}, \{\text{apple}\}.$$

The idea behind the apriori algorithm is to build up frequent itemsets starting with small sets.

1. First, we make a pass over the database and identify all the **frequent items** (frequent itemsets with only 1 item);
2. then we enumerate the 2−item sets by adding one element to each of the frequent 1−item sets, in all possible ways;
3. from these generated 2−item sets, only those that are frequent are retained.
4. This procedure is then repeated to obtain the frequent 3−item sets, and so on, until there are no new frequent itemsets on which to build.

Example Suppose that apple, bread, cheese, and watermelon are frequent items in a transaction dataset. Then we only need to consider 1−item sets selected from the family

$$L_1 = \{\text{apple}, \text{bread}, \text{cheese}, \text{watermelon}\}.$$

All other items can be ignored because no superset that contains them can be frequent according to the apriori property.

The candidates 2−item sets are thus:

$$\{\text{apple}, \text{bread}\}, \{\text{apple}, \text{cheese}\}, \{\text{apple}, \text{watermelon}\},$$
$$\{\text{bread}, \text{cheese}\}, \{\text{bread}, \text{watermelon}\}, \{\text{cheese}, \text{watermelon}\};$$

those that are frequent in that list are retained to form the family L_2, and so on. □

General Algorithm The general aprior algorithm is simple: let L_k denote the family of all frequent k−item sets. Form all possible $(k+1)$−item sets (the **candidates**) that contain a k−item set of L_k; those that are also frequent are placed into L_{k+1}.

Unfortunately, the apriori algorithm is unsuitable for data streams because the algorithm requires **multiple scans of the database**, and resources are often wasted on the **generation of candidate**, many of which ultimately ending up **discarded**, in typical applications.

A number of additional challenges also arise when mining frequent items in data streams:

- the data often must be processed in a **single pass**;
- computer memory is limited and cannot always store all of the stream's data, and
- frequent itemsets can be **time-sensitive** (concept drift).

Popular Algorithms In a streaming context, frequent itemset algorithms are classified based on a number of characteristics:

- **approximate** or **exact** (allows false positives/negatives or not)
- **per-batch** or **per-transaction** processing
- **incremental**, **sliding window**, or **adaptive**
- **frequent**, **closed**, or **maximal** itemsets

Some popular algorithms are shown in the table below (see [18] for a more comprehensive table).

Algorithm	Window	Batch?	Accuracy	Itemsets
LossyCounting	Landmark	Yes	False+	All
FP-stream	Tilted	Yes	False+	All
FDPM-1	Landmark	Yes	False-	All
MOMENT	Tilted	Yes	Exact	Closed
IncMine	Tilted	Yes	False+	Closed
estDec+	Landmark/ Damped	No	False+/-	Maximal

Table 28.6: Popular frequent itemset mining data streams algorithm.

LossyCounting Proposed in 2002 [21], **LossyCounting** is the first one-pass algorithm to find all frequent itemsets over a data stream. It does not allow false negatives and provides a **theoretical guarantee** on false positives.

For a user-defined **error parameter** $0 \leq \varepsilon \leq 1$ and **support threshold** $\theta \in (0, 1)$, all itemsets with a relative frequency in excess of θ will be found; conversely, no itemset whose true relative frequency is less than $\theta - \varepsilon$ will be included.

In a data stream, the LossyCounting algorithm for frequent 1−item sets works as follows:

1. the stream is divided into **buckets**, each of size $\lceil 1/\varepsilon \rceil$;
2. the buckets are **indexed sequentially** starting at 1;
3. counters of the form $\langle$`element`, `count`, `bucket_id`$\rangle$ are maintained, where `bucket_id` denotes the ID of the bucket being processed when the counter was created;
4. at the end of each bucket, we check if `count` + `bucket_id` $\leq \varepsilon N$, where N denotes the number of transactions seen so far – if the inequality holds, the counter is deleted.

The modification to handle frequent k−item sets is a bit more complicated. Assume that we have a **data structure** D, initially empty, consisting of a set of entries of the form (`itemset`, f, Δ) where f is the **estimated frequency** of the itemset and Δ is the **maximum possible error** in f.

1. The stream is divided into **buckets** of $w = \lceil 1/\varepsilon \rceil$ transactions each;
2. the buckets are **indexed sequentially** starting at 1
3. the current bucket id is denoted by `bcurrent`;

4. the transactions are not processed individually but in **batches**, each containing as many transactions as memory allows;
5. let β be the number of buckets in **main memory** in the batch being **currently processed** (β must be relatively "large");
6. D is updated as follows:
   a) for each entry (`itemset`, f, Δ) in D, increment f by the number of occurrences of `itemset` in the **current batch**;
   b) if $f + \Delta \leq$ `bcurrent`, the entry is **removed** from D;
   c) if an `itemset` has frequency f at least β times in the current batch and `itemset` does not occur in D, a new entry (`itemset`, f, `bcurrent` $- \beta$) is added to D.

We observe that a set `itemset` with true frequency at least εN must have an entry (`itemset`, f, Δ) in D. Furthermore, the true frequency is at least f and at most $f + \Delta$.

Note that we can also provide a list of itemsets with support threshold θ by outputting the entries in D with $f \geq (\theta - \varepsilon)N$.

Conceptually, LossyCounting is a rather simple algorithm, but its **efficient implementation** requires the use of **non-trivial data structures** whose descriptions are beyond the scope of these notes.[52]

52: A common implementation makes use of a "trie" (a **prefix tree**), which is a search tree for strings that can be dynamically updated. A full description of LossyCounting's implementation details can be found in [20].

FDPM-1 Unlike LossyCounting, the **frequent datastream pattern mining** algorithm FDPM-1 does not allow false positives, but it also has a high probability of finding **truly** frequent itemsets [32].

Users must specify parameters $0 \leq \delta \leq 1$ and $\theta \geq 1$ such that the resulting family P contains no itemset whose frequency is below the specified **support level** θ and includes any frequent $\theta-$item set with probability at least $1 - \delta$.[53]

53: FDPM-1 uses the **Chernoff bound** to achieve the probabilistic guarantee by modeling the appearance of an itemset in P as a **binomial random variable** – in each transaction, either the itemset appears or it does not. This requires the transactions to be **independent**, which is not necessarily valid in real applications.

$n \leftarrow 0$, $P \leftarrow \emptyset$;
while a new transaction t arrives **do**
  if $t \in P$ **then**
    increase t's count by 1;
  else
    if $|P| > n_0$ **then**
      calculate the running ϵ_n for the n observations;
      delete all entries in P that are potentially infrequent;
    end if
    insert t with an initial count 1 into P;
  end if
  $n \leftarrow n + 1$;
  output P on demand;
end while

Figure 28.6: Frequent datastream pattern mining FDPM-1, which produces the family P of frequent itemsets.

It can be shown that the **required number of transactions per batch** is

$$n_0 = 2(1 + \ln(2/\delta))/\theta;$$

the **running variable** is $\varepsilon_n = \sqrt{2\theta \ln(2/\delta)/n}$, where n is the number of transactions **seen so far**.

IncMine Proposed in 2008 [11], **IncMine** uses **sliding windows** to mine **frequent closed itemsets** (FCI), with a controlled number of false negatives. The algorithm can handle concept drift, has a high throughput, and only requires **modest** memory usage.

The sliding window looks at the w **most recent time units**.[54] The following illustration shows 4 time units: t_1, t_2, t_3, t_4, with 3, 2, 3, 4 transactions, respectively. Using $w = 2$, we have 3 **complete time windows** (t_1, t_2), (t_2, t_3), (t_3, t_4) over the 4 time units, containing 5, 5, 7 transactions, respectively.

54: The number of transactions arriving in each time unit may vary.

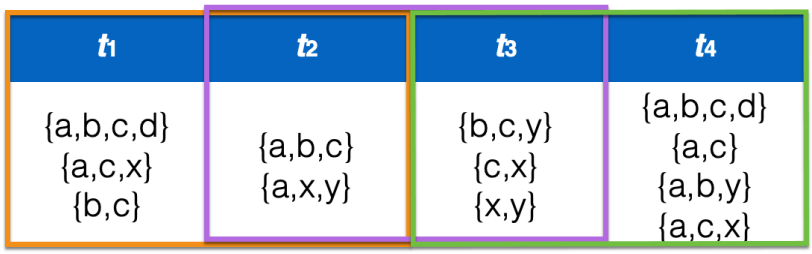


Figure 28.7: An illustration of a sliding window with $w = 2$ over 4 time units.

IncMine introduce a novel idea for controlling the number of itemsets that could eventually become frequent; the basic idea of **semi-FCI** is that an itemset whose first appearance in the window happens earlier than one with a later first appearance needs a **higher support** in order to be kept. Intuitively, if a **low support** itemset has been in the stream for a while, it is unlikely to become a **frequent** itemset.[55]

55: In other words, IncMine assessment of an itemset's support is **time-dependent**.

The set of semi-FCI is updated incrementally.[56] Briefly, consider the three semi-FCIs F, L, and C, where F is the set of semi-FCI over the **current time unit**, L is the set of semi-FCI over the **previous window**, and C is the set of semi-FCI over the **current window**. The incremental update task is to construct C by modifying L using F, dropping unpromising itemsets and adding new semi-FCI according to some protocol.

56: The details of the actual algorithm and its efficient implementation are rather technical and fall beyond our scope; details are available in [11, sec. 5].

Significantly faster processing times and smaller memory consumption have been reported, in comparison to other algorithms.[57] Of course, the No Free Lunch theorems are still in play, so *caveat emptor*.

57: IncMine was able to process up to 40,000 transactions per second for the t10i4 data stream (produced by a modified IBM data generator), outperforming LossyCounting and MOMENT by over two orders of magnitude. Memory consumption was found to be much below that of LossyCounting and MOMENT, and was relatively constant over a range of values of minimum support threshold. [11]

Mining Other Patterns Interesting patterns do not always appear as frequent subsets of a larger set. In a supermarket, for instance, we might be interested in tracking the **order** in which a customer picks up items (e.g., milk before eggs); these **precedence relations** might provide insight on merchandise layout, say. **Sequence mining** also has applications to bioinformatics, text mining, telecommunications, fraud detection, and cybersecurity.

Graph mining is another common pattern mining task; it subsumes itemset mining since itemsets can be modeled as a **complete subgraphs**. Notable graph mining algorithms include:

- **IncGraphMiner** (a variant of IncMine);
- **WinGraphMiner** (which maintains the frequent closed graphs in a fixed-size sliding window), and
- **AdaGraphMiner** (an extension of WinGraphMiner which can adapt to changes in the stream).

28.6 Examples

Various data streams mining notions discussed in this chapter are illustrated in the following examples, using R.

28.6.1 Obtaining Statistics

We got some practice doing batch analysis when working with the dailysales.csv dataset in Section 28.1.1. In this first example, we will work with the related weeklysales.csv dataset, which contains all the sales over a week.

First, we read in the data.

```
dfw <- read.csv("weeklysales.csv", stringsAsFactors = TRUE)
str(dfw)
head(dfw)
```

```
'data.frame': 154 obs. of  4 variables:
 $ sku   : Factor w/ 4 levels "A","B","C","D": 3 1 2 1 3 2 2 3 3 2 ...
 $ day   : Factor w/ 7 levels "Fri","Mon","Sat",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ time  : Factor w/ 112 levels "08:25","08:30",..: 25 28 30 34 35 37 43 46 50 56 ...
 $ amount: num  9.99 12.99 4.49 12.99 9.99 ...
```

Suppose that we are interested in the most popular item sold on Sunday.

We start by isolating the Sunday sales.

```
(sun <- dfw[dfw$day=='Sun',])
```

```
   sku day  time amount
1    C Sun 10:20   9.99
2    A Sun 10:28  12.99
3    B Sun 10:43   4.49
4    A Sun 11:17  12.99
5    C Sun 11:23   9.99
6    B Sun 11:30   4.49
7    B Sun 12:09   4.49
8    C Sun 12:22   9.99
9    C Sun 12:30   9.99
10   B Sun 13:00   4.49
11   C Sun 13:12   9.99
12   C Sun 13:20   9.99
13   B Sun 13:39   4.49
14   A Sun 16:11  12.99
15   D Sun 16:27  24.99
16   A Sun 16:35  12.99
17   A Sun 16:55  12.99
18   A Sun 16:59  12.99
```

Next, we only retain the sku column.

```
df2 <- sun['sku']
```

We find the summary count statistics as follows:

```
(ag <- aggregate(df2, by=list(item = df2$sku), FUN=length))
```

```
  item sku
1    A   6
2    B   5
3    C   6
4    D   1
```

We rename the columns for consistency's sake and exhibit the most popular item(s).

```
(agsun <- setNames(ag, c('sku','count')))
agsun[agsun$count==max(agsun$count),]
```

```
  sku count
1   A     6
3   C     6
```

28.6.2 Bloom Filter

In this example, we create a Bloom filter for a list of malware URLs excerpted from malwaredomainlist.com, found in `malwaredomainlist.csv`.

Recall that such a filter can produce false positives; that is, it can flag URL that do not appear on the list of URL used to create the filter.

We start by loading the data.[58]

58: This block of code could be replaced by a scraper that collects up-to-date information from a host file containing thousands of adware/malware URL into a list of strings ready for hashing.

```
badURLs <- read.csv('malwaredomainlist.csv')
head(badURLs)
```

```
                                                URL
1                          CPROHWIN0190.locaweb.com.br
2                          PEMLINWEB133.blacknight.com
3                    Static-IP-18150024815.cable.net.co
4 a184-50-238-184.deploy.static.akamaitechnologies.com
5                                  acf113.rev.netart.pl
6                                  acs169.rev.netart.pl
```

Next, we set the filter's parameters.

```
k <- 4      # Number of hash functions
m <- 100000 # Size of the filter array
```

The next stage is to build a "helper" function that creates a hash function from a sample URL (salt). The range of output values of the created hash function is $1, \ldots, m$ (inclusively). The `digest()` function (in package `digest`) creates hash digests of arbitrary `R` objects – in this case, URL strings.

```
makehash <- function ( salt, m ) {
    function( url ) {
        1 + strtoi(substr(digest::digest(paste(url,as.character(salt)),
                                         algo='xxhash32'), 1 , 5), 16L) %% m
    }
}
```

Then we create the k hash functions, by applying `makehash` to our sample URL k times.

```
f <- lapply( seq(1:k) , function(x) {makehash(x,m)})
```

For instance, we can apply the k hash functions to the sample URL `uottawa.ca`.

```
sampleURL = 'uottawa.ca'
for (i in 1:k) print(f[[i]](sampleURL))
```

```
[1] 17275
[1] 11876
[1] 29530
[1] 18068
```

We can now construct the Bloom filter from `badURLs`.

```
bf <- as.vector(rep(FALSE, m))
for (i in 1:k) {
    hashed <- lapply( badURLs$URL, f[[i]] )
    for (j in hashed) bf[j] = TRUE
}
```

The object `bf` is a logical vector of length m. In this example, we have the following distribution.

```
table(bf)
```

```
bf
FALSE  TRUE
97757  2243
```

We write a function that uses the Bloom filter to flag bad URL.

```
isBadURL <- function( url ) {
    rv <- TRUE
    for (i in 1:k) {
        hash <- f[[i]]( url )
        if (bf[hash] == FALSE) {
            rv <- FALSE
        }
      break
    }
    rv
}
```

We can now use the Bloom filter on a list of known safe sites obtained from moz.com/top500 (stored in `moz_dot_com_top500.csv`) to determine the percentage of false positives.

```
top <- read.csv('moz_dot_com_top500.csv')
(n <- dim(top)[1]) # number of rows
head(top)
```

```
[1] 500

URL
1  facebook.com
2   twitter.com
3    google.com
4   youtube.com
5 instagram.com
6  linkedin.com
```

We can compute the number of false positives in `top` as follows:

```
falsePositive <- 0
(c <- Reduce('+', lapply( top$URL, isBadURL)))
```

```
[1] 12
```

The proportion of false positives is thus quite low:

```
c / n
```

```
[1] 0.024
```

We can also determine which safe sites were considered "bad" by the Bloom filter built from the list of malware sites.

```
index = which(lapply( top$URL, isBadURL) == TRUE)
top[index,]
```

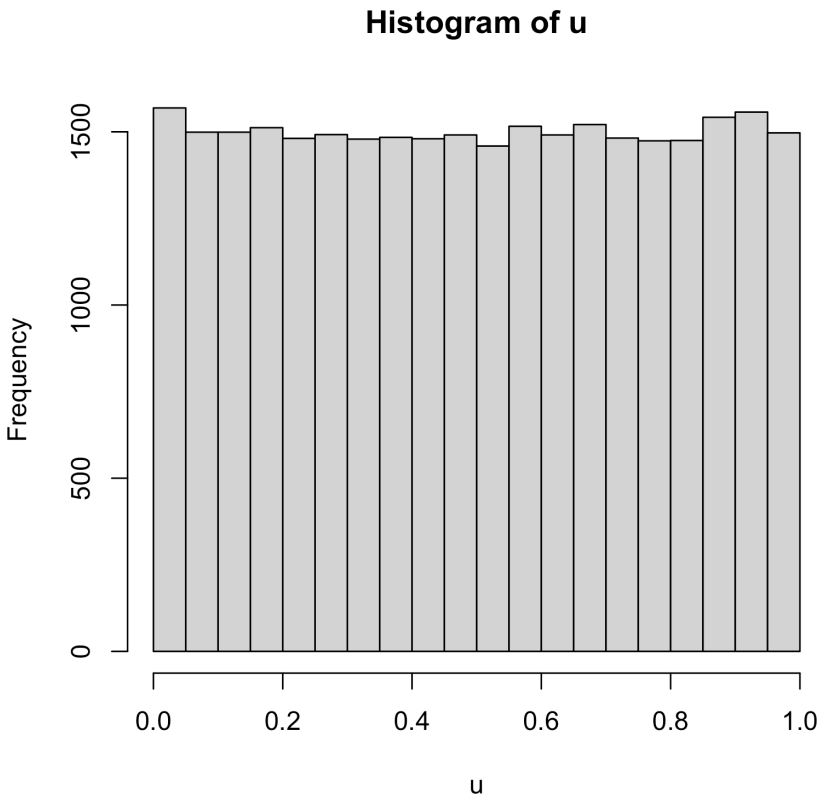
```
 [1] "wixsite.com"          "archive.org"
 [3] "time.com"             "constantcontact.com"
 [5] "webs.com"             "networksolutions.com"
 [7] "youku.com"            "yale.edu"
 [9] "loopia.com"           "mozilla.com"
[11] "allaboutcookies.org"  "dot.gov"
```

That is not great news for Yale University, not gonna lie...

28.6.3 Sampling With a Reservoir

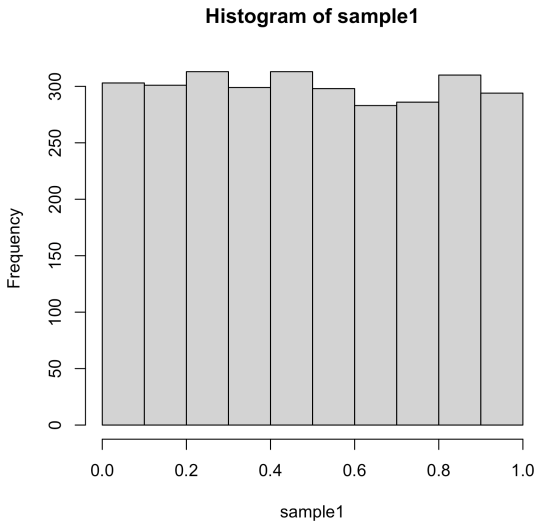
In the next two sections, we will compare sampling with reservoir and with hash. We will not be working with actual data streams, however. Instead, we will generate a long list of integers and process the numbers sequentially.

```
n <- 30000
set.seed(n) # for replicability
u <- runif( n, 0, 1 )  # uniform distribution
hist(u)
```



We see that the distribution is quite uniform. If we batch select 10% of the points of u uniformly randomly, we expect the sample distribution to also be uniform.

```
c <- as.integer(0.10 * n)
sample1 <- sample(u, c)
hist(sample1)
```



How would we implement sampling with a reservoir in R? First, we create the reservoir.

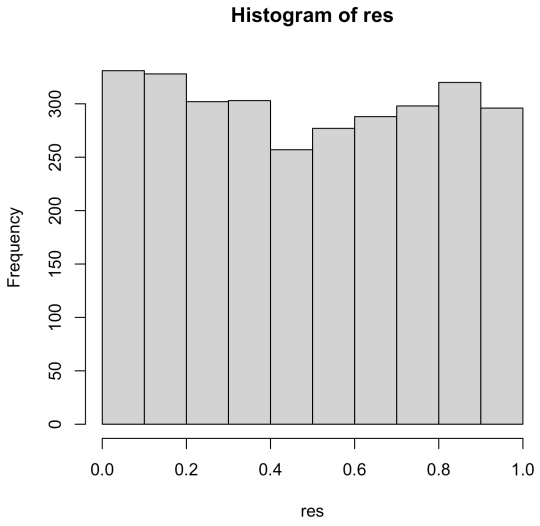
```
res <- u[1:c]
```

The reservoir sampling routine is shown below.

```
for (t in seq( c+1, length(u) )) {
    m <- sample( 1 , t , 1)
    if (m[1] <= c) {
        # select a random position in the reservoir
        # and replace with u[t]
        i <- sample( 1 , c , 1)
        u[i[1]] = u[t]
    }
}
```

Once again, the histogram shows a uniform distribution.

```
hist(res)
```



28.6.4 Sampling With a Hash Function

Sample with a hash function, we should once again expect to see a uniform sample distribution.

```
hashNumeric <- function (x) { 1 + strtoi(substr(digest::digest(as.character(x),
                                                               algo='xxhash32'),1,7),
                                         16L) %% n}
hashes <- setNames( as.data.frame(sapply(u, hashNumeric)) , c('hash') )
```

The `hashes` object is a vector with the same length as u.

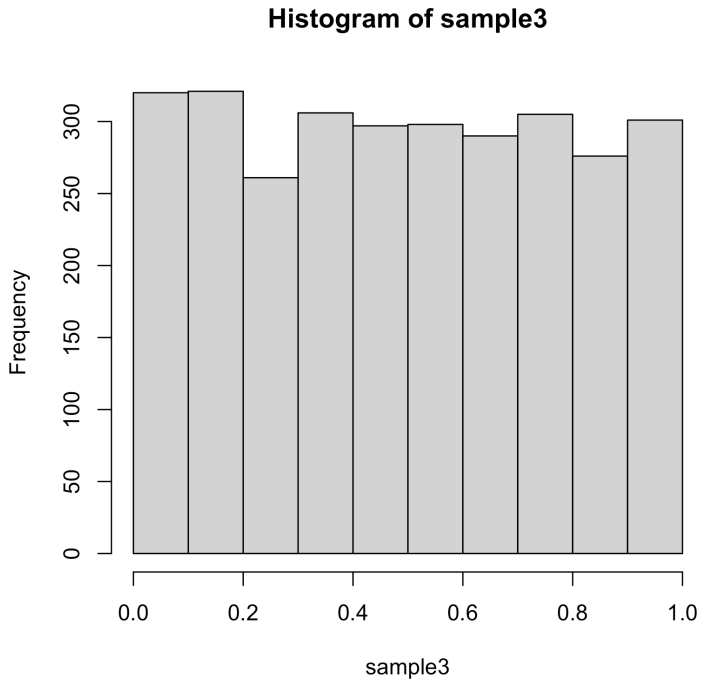
```
head(hashes)
```

```
   hash
1 28126
2 13495
3  1506
4 28238
5 15920
6 11666
```

The has sample may not necessarily contain exactly 3000 observations, however.

```
sample3 <- u[hashes$hash <= c]
length(sample3)
hist(sample3)
```

```
[1] 2975
```



28.6.5 Fading Window

We now turn to the simple fading window example discussed on pp.1828-1829 to identify recent most popular items. We start by setting the number of possible items in the data.

```
set.seed(2000) # for replicability
numItems <- 100
```

Next, we create a data "stream" of 10,000 sales.

```
p <- sort( runif( numItems-1 , 0 , 1 ))
r <- runif(5000,0,1)

findIndex <- function (x, d) {
    rv <- 1
    # Identity the interval with breakpoints given by d in which x lies
    for (i in 1:length(d)) {
        if (x > d[i]) { rv <- i+1 }
    }
    rv
}
```

The stream is built in such a way that popular items in the first half are not popular in the second half and vice versa.

```
items <- c(lapply(r, function(x) findIndex(x, p)),
           lapply(r, function(x) { numItems + 1 - findIndex(x, p) } ))
```

The top products in the first half of the stream are shown below.

```
firstHalf.df <- setNames(data.frame(unlist(items[1:length(items)/2])), c('item'))
ag1 <- setNames(aggregate(firstHalf.df, by=list(item = firstHalf.df$item), FUN=length),
                c('item', 'count'))
head(ag1[ order(-ag1$count),])
```

```
item count
  26   556
  50   550
  87   336
  90   294
  39   288
  20   286
```

The top products in the second half of the stream are shown below.

```
secondHalf.df <- setNames(data.frame(unlist(items[(length(items)/2+1):length(items)])),
                          c('item'))
ag2 <- setNames(aggregate(secondHalf.df, by=list(item = secondHalf.df$item), FUN=length),
                c('item', 'count'))
head(ag2[ order(-ag2$count),])
```

```
item count
  75   278
  51   275
  14   168
  11   147
  62   144
  81   143
```

We set up the parameters for the fading window algorithm.

```
t = 0.5  # threshold
c = 1/length(items)  # decaying factor
scores <- rep(0.0, numItems)
```

We now process the first half the data "stream" with the fading window algorithm.

```
for (i in 1:(length(items)/2)) {
    scores <- (1-c)*scores
    j <- items[[i]]
    scores[[j]] <- scores[[j]] + 1
}
```

The top scores are shown below.

```
scores.df <- setNames(data.frame(c(1:length(scores)),unlist(scores)), c('item','score'))
head(scores.df[ order(-scores.df$score),])
```

```
item    score
  26 219.6582
  50 215.2115
  87 132.1868
  90 114.2575
  20 112.9336
  39 111.5845
```

The same items appear, although the order of the 5th and 6th items has been switched. And of course, we can do the same for the second half of the "stream".

```
scores <- rep(0.0, numItems)

for (i in (length(items)/2+1):length(items)) {
    scores <- (1-c)*scores
    j <- items[[i]]
    scores[[j]] <- scores[[j]] + 1
}

scores.df <- setNames(data.frame(c(1:length(scores)),unlist(scores)), c('item','score'))
head(scores.df[ order(-scores.df$score),])
```

```
item    score
  75 219.6582
  51 215.2115
  14 132.1868
  11 114.2575
  81 112.9336
  62 111.5845
```

The same items appear, although the order of the 5th and 6th items has been switched.

28.6.6 Adaptive Sliding Window Algorithm

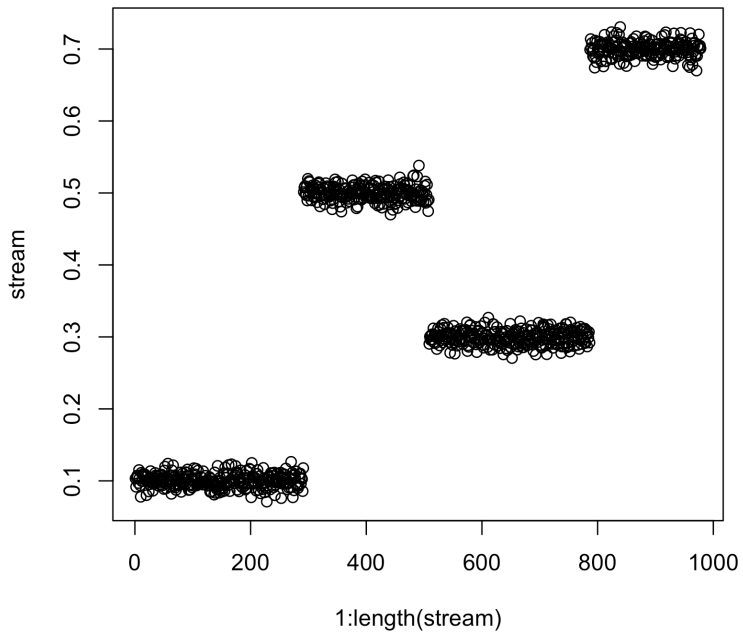
In this example, we play with the ADWIN algorithm of Section 28.2.2.

First, we create a random data "stream" with $n = 4$ concepts.

```
set.seed(0) # for replicability
nConcepts <- 4
minLen <- 150
maxLen <- 300
noiseLevel <- 0.01
len <- sample(minLen:maxLen, nConcepts)
means <- c(0.1, 0.5, 0.3, 0.7)
stream <- list()
for (i in 1:nConcepts) {
    t <- rep(means[[i]], len[[i]]) + noiseLevel * rnorm(len[[i]], 0, 1)
    stream <- c(stream, t)
}
```

We display the complete stream below.

```
plot( 1:length(stream), stream )
```



We can obtain the empirical mean of the stream along each concept.

```
means.2 <- c()
cumlen = cumsum(unlist(len))
means.2[1] <- mean(unlist(stream[1:cumlen[1]]))

for(t in 2:nConcepts){
        means.2[t] <- mean(unlist(
                        stream[cumlen[t-1]+1:cumlen[t]])
                       )
    }
means.2
```

```
[1] 0.1002887 0.3952116 0.4630066 0.7000306
```

Let's see if ADWIN can detect the different concept means. First, we implement the threshold function: $\varepsilon(W) = \sqrt{\frac{1}{2m} \ln \frac{4|W|}{\delta}}$ where W is a list of numbers (the variable-length window) and $m = \frac{2}{1/|W_0|+1/|W_1|}$. In the code below, i represents the splitting indices.

```
delta = 0.5 # A value between 0 and 1; a value of 0.5
            # favours neither FP nor FN

threshold <- function (W, i) {
    W0 <- W[1:i]
    W1 <- W[(i+1):length(W)]
    m <- 2/(1/length(W0)+1/length(W1))
    eps <- sqrt(1/(2*m)*log(4*length(W))/delta)
    eps
}
```

Next, we Initialize the window W to contain only the first observation in the stream.[59]

59: Normally, one should consider a sizeable initial chunk of the stream, as in the exercises.

```
W <- stream[1]
border <- 0

for (t in 2:length(stream)) {
    W <- c(unlist(W), stream[[t]])

    if (length(W) > 1) {
        i <- 0
        repeat {
            i <- i+1
            if (i >= length(W)) break;

            u0 <- mean(W[1:i])
            u1 <- mean(W[(i+1):length(W)])
            thd <- threshold(W, i)
            if (abs(u0 - u1) > thd) {
                border = border + i
```

```
                print(paste('Detected new mean:',
                    as.character(u1),
                    'between indices', border,
                    'and', border+1))
                W <- W[(i+1):length(W)]
                i <- 0
            }
        }
    }
}
```

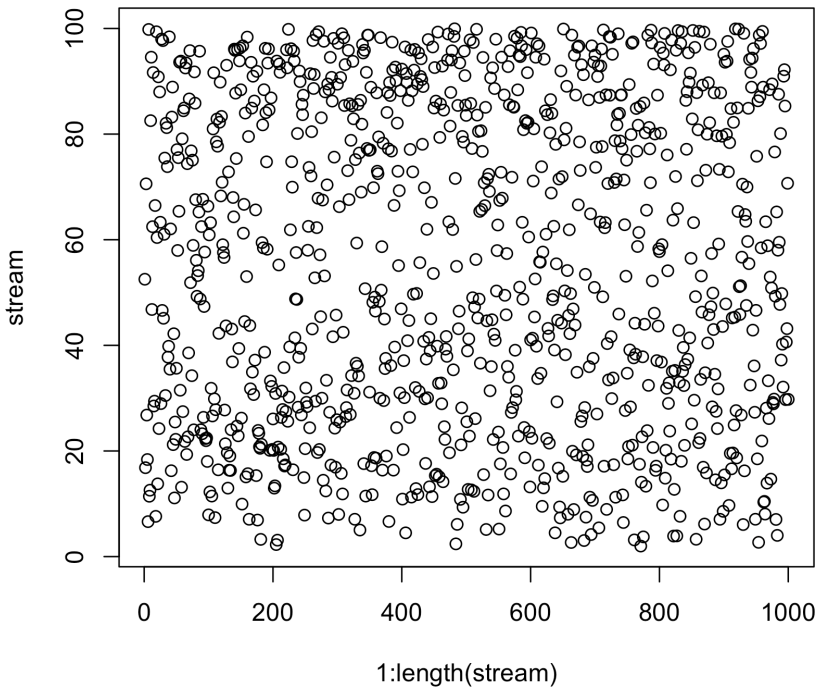
```
[1] "Detected new mean: 0.502436993351689 between indices 291 and 292"
[1] "Detected new mean: 0.299156929106440 between indices 508 and 509"
[1] "Detected new mean: 0.697717000444021 between indices 786 and 787"
```

28.6.7 Partition Incremental Discretization Algorithm

In this example, we study the mechanics of PID (of pp.1836-1838).

We first simulate a data stream of 1000 random values drawn from 3 beta distributions.

```
set.seed(1) # for replicability
n <- 1000
maxVal <- 100
stream <- sample( c(rbeta(2*n, 2, 5),
                    rbeta(2*n, 5, 1),
                    rbeta(n, 2, 2)), 1000 ,
                  replace = FALSE) * maxVal

plot( 1:length(stream), stream )
```



We construct Layer 1 from the simulated stream; for simplicity, we use intervals with integer breakpoints of width 1.[60]

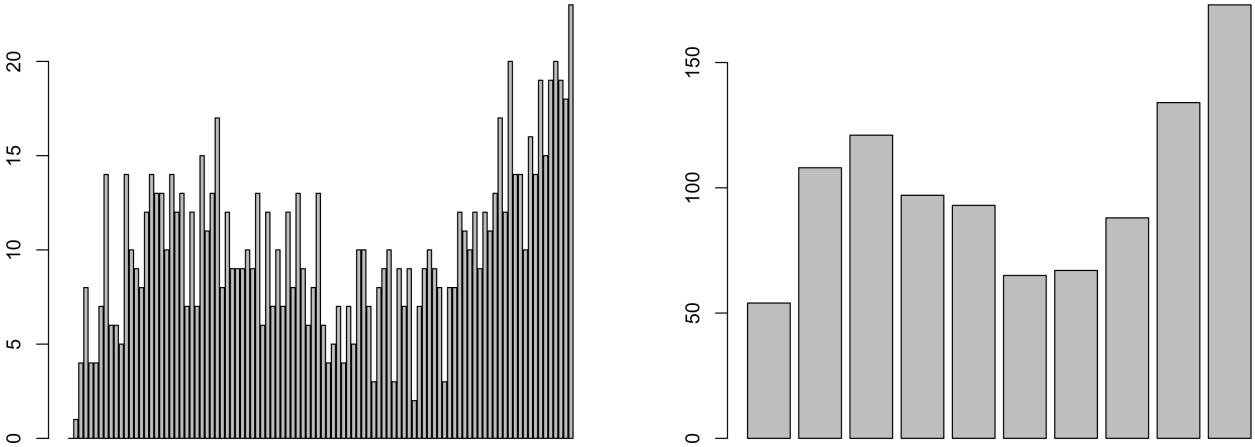
60: We could also use the split operator.

```
bins1 <- rep(0, maxVal)

for (i in 1:length(stream)) {
    # Increment the count for the bin in which the ith
    # element of stream falls
    b <- ceiling(stream[[i]])
    bins1[[b]] <- bins1[[b]] + 1
}
```

Next, we construct a $k = 10$ equal-width Layer 2 histogram from Layer 1's bin counts.[61]

61: Obtaining an equal-frequency histogram is left as an exercise.

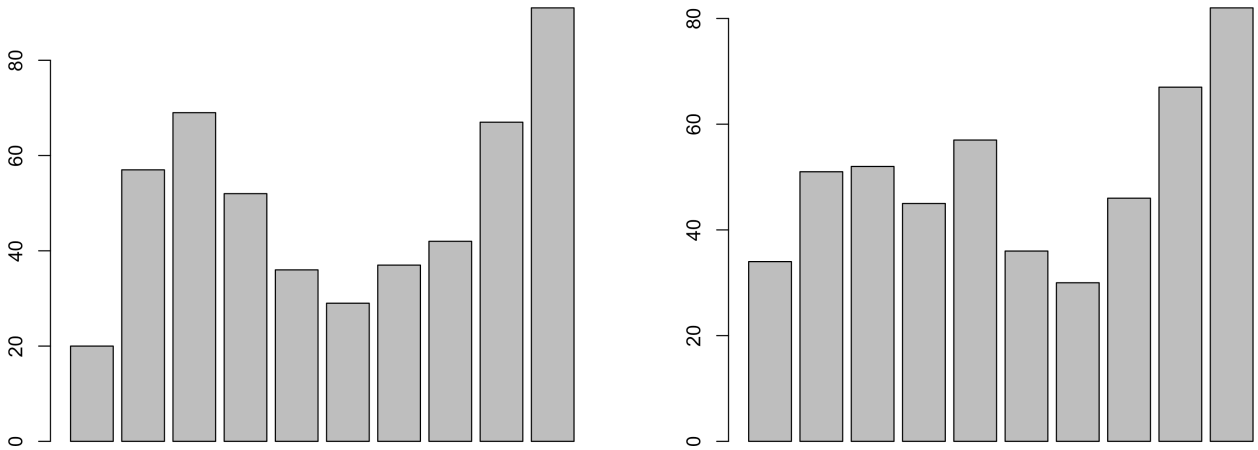
```
k <- 10
width <- ceiling (maxVal / k)
l2bins <- rep(0, k)

for (i in 1:k) {
    lb <- (i-1)*width + 1
    ub <- i*width
    for (j in lb:ub) {
        l2bins[[i]] <- l2bins[[i]] + bins[[j]]
    }
}

barplot(bins1); barplot(l2bins)
```



28.6.8 Histogram Drift

Using the same stream as in the previous section, we explore if there is a difference between the histogram for either halves of the stream.

```
bins1 <- rep(0, maxVal); bins2 <- rep(0, maxVal)

# Layer 1 - first half
for (i in 1:(length(stream)/2)) {
    # Increment the count for the bin in which the ith element of stream falls
    b <- ceiling(stream[[i]])
    bins1[[b]] <- bins1[[b]] + 1
}

# Layer 1 - second half
for (i in (length(stream)/2+1):length(stream)) {
    b <- ceiling(stream[[i]])
    bins2[[b]] <- bins2[[b]] + 1
}

# Layer 2 - both halves simultaneously
l2bins1 <- rep(0, k); l2bins2 <- rep(0, k)

for (i in 1:k) {
    lb <- (i-1)*width + 1
    ub <- i*width
    for (j in lb:ub) {
        l2bins1[[i]] <- l2bins1[[i]] + bins1[[j]]
        l2bins2[[i]] <- l2bins2[[i]] + bins2[[j]]
    }
}

barplot(l2bins1); barplot(l2bins2)
```



What about it? Are the two half-stream histograms similar or different?

28.7 Exercises

1. Find the top two items sold between 1pm and 5pm in `dailsales.csv` (note the time format in the dataset).
2. Find the most popular and the least popular items sold on each day of the week in `weeklysales.csv`.
3. Consider the Bloom filter of Section 28.6.2. Change the values of k and m to see how the rate of false positives changes. Do the results change significantly if we use updated lists of malware and safe sites? Implement your own hash functions and compare the results with those using the given hash function.
4. Recreate the examples from 28.6.3 and 28.6.4 using other distributions to generate u (such as `rnorm()`, etc.). Implement your own hash functions and compare the results with those using the given hash function.
5. We revisit the fading window example 28.6.5. How did the results from the fading window algorithm over the entire "stream" compare with the actual top items for the two halves? Apply the fading window algorithm to the entire stream instead; what can be said about the results? If we adjust the value of c (e.g., $c = 0.01$), `numItems`, and/or the number of transactions in the data "stream"; how do the results change?
6. In the ADWIN example of Section 28.6.6, instead of hardcoding the means, generate random means and see how the algorithm performs. Experiment with different values for `nConcepts`, `minLen`, `maxLen`, and `noiseLevel`. Do we encounter false positives? False negatives? As it stands, the ADWIN code above does *not* show the mean of the initial portion of the data "stream". Modify the code so that it does show the first mean.
7. In the PID example of Section 28.6.7, construct an equal-frequency Layer 2 histogram from Layer 1; experiment with different values of k, `maxVal`, and choices of distributions. Write a function that takes Layer 1 bin counts and other appropriate arguments and outputs Layer 2 bin counts.
8. Use a suitable statistical test on the two sets of bin counts to see if the histograms of Section 28.6.8 are different, with statistical significance.
9. Consider the dataset `flights1_2019_1.csv`.
    a) Transform the dataset so that the observational units are the airports and date. Create variables as needed. Assume that each new day's observations constitute a new batch. Does the stream exhibit concept drift? Maintain and display statistics and histograms for some of the variables in the stream.
    b) Conduct a streaming $k-$means clustering of the data. Play with values of k and α. Conduct a streaming two-phase clustering of the data, under the same conditions as above. Play with parameter values in CluStream and DenStream.[†] Comment on the results.

Chapter References

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

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

[3] C.C. Aggarwal et al. 'A framework for clustering evolving data streams'. In: *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*. 2003, pp. 81–92.

[4] C.C. Aggarwal et al. 'A framework for on-demand classification of evolving data streams'. In: *IEEE Trans. Knowl. Data Eng.* 18 (May 2006), pp. 577–589.

[5] M. Basseville, I.V. Nikiforov, et al. *Detection of Abrupt Changes: Theory and Application*. Vol. 104. Prentice Hall Englewood Cliffs, 1993.

[6] A. Bifet and R. Gavaldà. 'Adaptive learning from evolving data streams'. In: *Advances in Intelligent Data Analysis VIII*. Ed. by Niall M. Adams et al. Springer Berlin Heidelberg, 2009, pp. 249–260.

[7] A. Bifet et al. *Machine Learning for Data Streams: with Practical Examples in MOA*. Adaptive Computation and Machine Learning series. MIT Press, 2018.

[†] You may need to consult appropriate references for the algorithm details and implementation.

[8] P. Boily. *Data Understanding, Data Analysis, and Data Science (Course Notes)*, *volumes 1-5*. QED/Idlewyld, 2026.

[9] P. Boily, S. Davies, and J. Schellinck. *The Practice of Data Visualization*. Data Action Lab, 2023.

[10] F. Cao et al. 'Density-based clustering over an evolving data stream with noise'. In: *Proceedings of the 2006 SIAM International Conference on Data Mining (SDM)*, pp. 328–339.

[11] J. Cheng, Y. Ke, and W. Ng. 'Maintaining frequent closed itemsets over a sliding window'. In: *J. Intell. Inf. Syst.* 31.3 (Dec. 2008), pp. 191–215.

[12] P. Domingos and G. Hulten. 'Mining high-speed data streams'. In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2000, pp. 71–80.

[13] B. Dykes. 'Big Data: Forget Volume and Variety, Focus On Velocity'. In: *Forbes* (June 2017).

[14] R. El Sibai et al. 'An in-depth analysis of CUSUM algorithm for the detection of mean and variability deviation in time series'. In: *Web and Wireless Geographical Information Systems*. Ed. by M. R. Luaces and F. Karimipour. Springer, 2018, pp. 25–40.

[15] J. Gama. *Knowledge Discovery from Data Streams*. Chapman & Hall/CRC Data Mining and Knowledge Discovery Series. CRC Press, 2010.

[16] V. Kanaujia and S. Chakraborty. 'Exploring probabilistic data structures: Bloom filters'. In: *OpenSourceForU* (May 2018).

[17] M. Lazarescu, S. Venkatesh, and H. Bui. 'Using multiple windows to track concept drift'. In: *Intell. Data Anal.* 8 (Mar. 2004), pp. 29–59.

[18] V.E. Lee, R. Jin, and G. Agrawal. 'Frequent pattern mining in data streams'. In: *Frequent Pattern Mining*. Ed. by Charu C. Aggarwal. Springer International Publishing, 2007, pp. 199–224.

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

[20] G.S. Manku. 'Frequent Itemset Mining over Data Streams'. In: *Data Stream Management - Processing High-Speed Data Streams*. Ed. by Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Data-Centric Systems and Applications. Springer, 2016, pp. 209–219.

[21] G.S. Manku and R. Motwani. 'Approximate frequency counts over data streams'. In: *Proceedings of the 28th International Conference on Very Large Data Bases*. VLDB Endowment, 2002, pp. 346–357.

[22] M. Miner and A. Shook. *MapReduce Design Patterns*. O'Reilly Media, Inc., 2012.

[23] M. Stirling et al. 'Concept drift detector selection for Hoeffding adaptive trees'. In: *AI 2018: Advances in Artificial Intelligence*. Ed. by Tanja Mitrovic, Bing Xue, and Xiaodong Li. Springer International Publishing, 2018, pp. 730–736.

[24] W.N. Street and Y.S. Kim. 'A streaming ensemble algorithm (SEA) for large-scale classification'. In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2001, pp. 377–382.

[25] J. Tenenbaum. *Computational Cognitive Science Group at MIT*.

[26] H. Wang et al. 'Mining concept-drifting data streams using ensemble classifiers'. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2003, pp. 226–235.

[27] G. Widmer and M. Kubat. 'Learning in the presence of concept drift and hidden contexts'. In: *Machine Learning* 23.1 (1996), pp. 69–101.

[28] Wikipedia. *Bloom filter*.

[29] Wikipedia. *Kolmogorov-Smirnov test*.

[30] Wikipedia. *Kullback-Leibler divergence*.

[31] Wikipedia. *Online algorithm*.

[32] Jeffery Xu Yu et al. 'False positive or false negative: mining frequent itemsets from high speed transactional data streams'. In: *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*. VLDB Endowment, 2004, pp. 204–215.

Data Understanding, Data Analysis, and Data Science

Volume 4 – Techniques of Data Analysis

Advance your analytical skillset with **Techniques of Data Analysis**, the fourth volume in the **Data Understanding, Data Analysis, and Data Science** series. This volume explores specialized topics that push the boundaries of standard methods, offering insight into how data science adapts to complex and dynamic challenges.

Chapters cover queueing systems, Bayesian analysis, anomaly detection, text mining, sentiment analysis, and data stream mining. Each topic is presented with an emphasis on conceptual clarity and supported by practical examples that demonstrate the diversity of real-world applications.

By extending the tool-agnostic and understanding-first philosophy of the series, this volume prepares readers to approach advanced techniques with flexibility and purpose. Whether read in tandem with instruction or as a self-directed resource, **Techniques of Data Analysis** is a guide to the nuanced methods that enrich modern data science.

About the Author

Patrick Boily is an Assistant Professor in the Department of Mathematics and Statistics at the University of Ottawa. He earned his Ph.D. in Mathematics in 2006 and is the author of seven textbooks on mathematics, statistics, and data science, available at idlewyldanalytics.com .

Since 1999, he has taught more than 75 courses at the University of Ottawa, the Université du Québec en Outaouais, and Carleton University. From 2008 to 2012, he served as a federal public servant, contributing to several projects including the award-winning Canadian Vehicle Use Study. From 2012 to 2019, he launched and managed Carleton University's Centre for Quantitative Analysis and Decision Support (CQADS), and he is a founding member of the Data Action Lab, which offers workshops, short courses, and consulting services in data analysis.

Patrick's academic work focuses on the application of mathematics and statistics to evidence-based decision support. He has provided consulting services to a wide range of public and non-profit organizations, including United Way, the Public Health Agency of Canada, the Canadian Air Transport Security Authority, and the Department of National Defence. His areas of expertise include operations research, data science and predictive analytics, stochastic modelling, and simulation.

Patrick is an avid hockey player, cross-country skier, cyclist, mountain biker, and swimmer; he enjoys crosswords, playing the guitar, and watching British murder mysteries. He lives with his family in Wakefield, Quebec.

IDLEWYLD

QUADRANGLE