Data Mining and Discovery Lecture Notes

J.D.P. Evans

September 11, 2023

Contents

Part I

Introduction

Chapter 1

Overview

1.1 What is Data Mining?

The obvious question that we must begin with is as follows: what is data mining? While this is poorly defined in general, roughly data mining is the process of automating the procedure of discovering useful information in large data sets. The goal is to find novel and useful patterns that might otherwise remain unknown, as well as potentially predicting the outcome of future observations.

Due to the size of the sets in question, there is an obvious need to automate this process. Consequently, its approach can be quite different from traditional data scientific strategies. As such, we differentiate between data mining and other information discovery tasks. For example, queries such as looking up individual records in a database or finding web pages that contain a particular set of keywords are not considered data mining tasks. This is because such tasks can be accomplished through simple interactions with a database management system or an information retrieval system. These rely on traditional computer science techniques such as indexing structures, query processing algorithms etc. Nonetheless, data mining can be used to enhance the performance of these systems.

Often, *Knowledge Discovery* and *Data Mining* are terms which are used interchangeably. There are also many other terms used alongside, such as knowledge extraction, information discovery, exploratory data analysis, information harvesting and unsupervised pattern recognition. More recently there has been a trend towards treating Knowledge Discovery as the entire knowledge extraction process, of which data mining is but one part. This is the view we will take and so, with this in mind, we say that the entire *Knowledge Discovery in Databases (KDD)* process (see Figure 1.1) involves converting raw data into useful information. The first step is to input some data which can be stored in a variety of formats (flat files, spreadsheets or relational tables) and may reside in a centralised data repository, or be distributed across multiple sites. If we are to be really exact, we should probably include an earlier step of selecting the data before input. Next, the *preprocessing* step transforms this raw input data into an appropriate format for subsequent analysis. This can include combining data from multiple sources, cleaning data to remove noise or selecting records and features that are relevant to the data mining task at hand. Data can often be from different sources and so data needs to be transformed into a common format (e.g. ensuring measure-

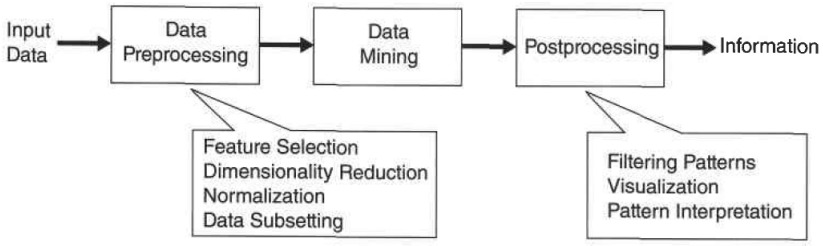


Figure 1.1: The process of KDD [2]

ments are all in the same units, such as metres). Because of the ways data can be collected and stored, this is usually the most laborious and time-consuming step. Next, the actual data mining takes place and the results are integrated into decision support systems. This is where *postprocessing* takes place and is used to ensure that only valid and useful results are incorporated into the support system. This could include visualisation or hypothesis testing methods. A key feature at this step is how the results will be interpreted. The results could be utterly breathtaking, but if they cannot reach their target audience, you are facing an uphill battle. See Chapter 10 of [2] for a discussion of this.

1.2 Why Data Mining?

Now that we have an idea of what data mining actually is (even if we have no clue how to go about it), it may be worthwhile discussing some motivations. Rapid advances in data collection and storage technology, coupled with the ease with which data can be generated and disseminated, have triggered an explosive growth of data. This is often referred to as *big data*. Such data can be from almost any area, be it business and industry, science and engineering, medicine and biotechnology, government, and so on. However, with data sets with sizes of terabytes, petabytes or even exabytes (volume), not to mention the complexity (variety) and rate at which it is collected (velocity), it has become too great a task for humans to analyse unaided. Thus, we need automated tools for information extraction. We start with some examples.

- Business and Industry

Point-of-sale data collection (barcode scanners, smart card technology etc.) have allowed retailers to collect up-to-the-minute data about customer purchases. Retailers can utilise this information, along with other business-critical data, to help better understand customer needs and spending patterns. This could help with customer profiling, targeted marketing, store layout, fraud detection and automated buying and selling (e.g. high-speed stock trading where decisions take place in less than a second). It can also help answer questions such as who the most profitable customers are, or which products can be cross-sold/up-sold. To answer these questions, data mining techniques such as *associated analysis* have been developed. This could be an area you can explore in your report.

Another angle to this is the continued growth of the internet. As this has continued to revolutionise the way we interact and make decisions, companies have begun exploring ways to exploit this. For example, data about our online viewing or shopping preferences can be used to provide personalised recommendations of products. On a more negative note, such methods can and are used to spread disinformation. Data mining also plays a prominent role in supporting other Internet-based services, such as filtering spam messages, answering search queries, social update suggestions etc. The large corpus of text, images and videos available on the Internet has enabled a number of advancements in data mining methods, including deep learning. This is another area which can be explored in your report.

- Medicine, Science and Engineering

Researchers are rapidly accumulating data that is key to significant new discoveries. For example, an important step towards improving our understanding of the Earth's climate system has been the deployment by NASA of a series of Earth orbiting satellites that continuously generate global observations of the land surface, oceans and atmosphere. Because of the size and spatio-temporal nature of the data, traditional methods are often unsuitable to analyse such data sets.

Another example is in molecular biology where large amounts of genomic data has been gathered in the hope of better understanding the structure and function of genes. In the past, traditional methods allowed scientists to study only a few genes at a time in a given experiment. More recently, breakthroughs in microarray technology have enabled scientists to compare the behaviour of thousands of genes under various situations. Such comparisons can help determine the function of each gene, and perhaps isolate the genes responsible for certain diseases. However, the noisy, high-dimensional nature of data requires new data analysis methods. In addition to analysing gene expression data, data mining can also be used to address other important biological challenges such as protein structure prediction, multiple sequence alignment, the modelling of biochemical pathways and phylogenetics.

Based on the above, we can outline some general motivating challenges that we would like to tackle.

1. Scalability

   If data mining algorithms are to handle massive data sets, they must be scalable. Many data mining algorithms employ special search strategies to handle exponential search problems. Scalability may also require the implementation of novel data structures to access individual records in an efficient manner. For a general overview of techniques for scaling up data mining algorithms, see Appendix F of [2].

2. High Dimensionality

   It is now common to encounter data sets with hundreds or thousands of attributes instead of the handful common a few decades ago. In bioinformatics, gene expression data often involves thousands of features. Data sets with temporal or spatial components also tend to have high dimensionality. For example, consider a data set that contains measurements of temperature at various locations. If the temperature

measurements are taken repeatedly over an extended period of time, the number of dimensions (features) increases in proportion to the number of measurements taken.

Traditional data analysis techniques that were developed for low-dimensional data often do not work well for such high-dimensional data due to issues such as *the curse of dimensionality*. Furthermore, for some data analysis algorithms, the computational complexity increases rapidly as the dimensionality (the number of features) increases.

3. Heterogeneous and Complex Data

   Traditional data analysis methods often deal with data sets containing attributes of the same type, either continuous or categorical. As the role of data mining has grown, so has the need for techniques that can handle heterogeneous attributes. Recent years have also seen the emergence of more complex data objects, such as those arising from web and social media data. These contain text, hyperlinks, images, audio and videos. Another example is DNA data with sequential and three-dimensional structure. One final example is climate data consisting of measurements (temperature, pressure etc.) at various times and locations on the Earth's surface. Techniques developed for mining such complex objects should take into consideration relationships in the data, such as temporal and spatial autocorrelation, graph connectivity, and parent-child relationships between the elements in semi-structured text and XML documents.

4. Data Ownership and Distribution

   Sometimes, the data needed for an analysis is not stored in one location or owned by one organisation. Instead, the data are geographically distributed among resources belonging to multiple entities. This requires the development of distributed data mining techniques. The challenges faced by distributed data mining algorithms include the following:

   - how to reduce the amount of communication needed to perform the distributed computation;
   - how to effectively consolidate the data mining results obtained from multiple sources; and
   - how to address data security and privacy issues.

5. Non-traditional Analysis

   The traditional statistical approach is based on a hypothesis/test paradigm, i.e. a hypothesis is proposed, an experiment designed to gather the data and then the data are analysed with respect to the hypothesis. This process is extremely labour intensive and current data analysis tasks often require the generation and evaluation of thousands of hypotheses. Consequently, the development of some data mining techniques has been motivated by the desire to automate the process of hypothesis generation and evaluation. Moreover, data need not be the result of a carefully designed experiment. Instead, the data may be opportunistic samples of the data, rather than random samples.

1.3 Data Mining Tasks

Roughly speaking, we can divide data mining into two major categories:

1. Predictive tasks

   The objective here is to predict the value of a particular attribute based on the values of other attributes.

   - The *target/dependent variable* is what we call the attribute to be predicted.
   - The *explanatory or independent variables* are what we call the attributes used for making the prediction.

2. Descriptive tasks

   This time, the objective is to derive patterns (correlations, trends, clusters, trajectories and anomalies) that summarise the underlying relationships in the data. These tasks are often exploratory in nature and frequently require postprocessing techniques to validate and explain the results.

Within these two rough categories, there exist four core data mining tasks:

- Classification.
- Association Analysis.
- Cluster Analysis.
- Anomaly Detection.

In these lecture notes, we will explore two of these tasks, though depending on your report topic, you may end up exploring a third of these core tasks. For the remainder for this section, we will briefly discuss each of these four tasks.

- Predictive Modelling.

The idea here is to create a model for the target variable as a function of the explanatory variables. There are two types of predictive modelling task: *classification* (used for discrete target variables) and *regression* (used for continuous target variables).

e.g. Predicting whether a web user will make a purchase at an online bookstore is a classification task since the target variable is binary-valued.

e.g. Forecasting the future price of a stock is a regression task because price is a continuous valued attribute.

In both cases, the goal is to learn a model that minimises error between the predicted and true values of the target variable.

Let's look at a more detailed example. Consider the task of predicting a species of flower based on the characteristics of the flower. In particular, consider classifying an Iris flower as one of the following three Iris species: Setosa, Versicolour, or Virginica. To perform this task

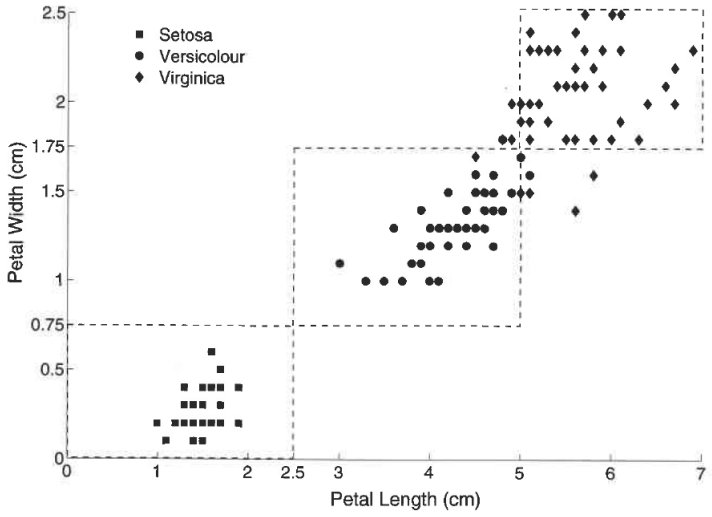


Figure 1.2: Petal width versus petal length for 150 Iris flowers, [2].

we need a data set containing the characteristics of various flowers of these three species. A data set with this type of information is the well-known Iris data set from the UCI Machine Learning Repository at `https://archive.ics.uci.edu/ml/datasets/iris`. In addition to the species of a flower, this data set contains four other attributes: sepal width, sepal length, petal length and petal width. Figure 1.3 shows a plot of petal width versus petal length for the 150 flowers in the Iris data set.

Note that petal width is broken into the categories *low, medium* and *high*, which corresponds to the intervals [0, 0.75), [0, 75, 1.75), [1.75, ∞), respectively. Likewise, petal length is broken into the categories *low, medium* and *high*, which correspond to the intervals [0, 2.5), [2.5, 5), [5, ∞), respectively. Based on these three categories of petal width and length, the following rules can be derived:

- Petal width low and petal length low implies Setosa.

- Petal width medium and petal length medium implies Versicolour.

- Petal width high and petal length high implies Virginica.

While these rules do not classify all of the flowers, they do a good (but not perfect) job of classifying most of the flowers.

N.B. The flowers from the Setosa species are well separated from the Versicolour and Virginica species with respect to petal width and length, but the latter two species overlap somewhat with respect to these attributes.

- Association Analysis

This is used to discover patterns that describe strongly associated features in the data. The discovered patterns are typically represented in the form of implication rules or feature

subsets. Because of the exponential size of its search space, the goal of association analysis is to extract the most interesting patterns in an efficient manner. This could be used to find groups of genes that have related functionality, identifying web pages that are accessed together, or understanding the relationships between different elements in Earth's climate system.

- Cluster Analysis

This seeks to find groups of closely related observations so that observations which belong to the same cluster are 'more similar' to each other than observations which belong to other clusters. Clustering can be used to group sets of related customers, find areas of the ocean that have a significant impact on the Earth's climate, and compress data.

For example, consider the following collection of news articles:

Collection of News Articles	
Article	Word-frequency pairs
1	dollar: 1, industry: 4, country: 2, loan: 3, deal: 2, government: 2
2	machinery: 2, labour: 3, market: 4, industry: 2, work: 3, country: 1
3	job: 5, inflation: 3, rise: 2, jobless: 2, market: 3, country: 2, index: 3
4	domestic: 3, forecast: 2, gain: 1, market: 2, sale: 3, price: 2
5	patient: 4, symptom: 2, drug: 3, health: 2, clinic: 2, doctor: 2
6	pharmaceutical: 2, company: 3, drug: 2, vaccine: 1, flu: 3
7	death: 2, cancer: 4, drug: 3, public: 4, health: 3, director: 2
8	medical: 2, cost: 3, increase: 2, patient: 2, health: 3, care: 1

These news articles can be grouped based on their respective topics. Each article is represented as a set of word-frequency pairs $(w : c)$, where w is a word and c is the number of times the word appears in the article.

There are two natural clusters in the data set. The first cluster consists of the first four articles, which correspond to news about the economy, while the second cluster contains the last four articles, which correspond to news about healthcare. A good clustering algorithm should be able to identify these two clusters based on the similarity between words that appear in the articles.

- Anomaly Detection

This is the task of identifying observations whose characteristics are significantly different from the rest of the data. Such observations are known as anomalies or outliers. The goal of an anomaly detection algorithm is to discover the real anomalies and avoid falsely labelling normal objects as anomalous. In other words, a good anomaly detector must have a high detection rate and a low false alarm rate. Applications of anomaly detection include the detection of fraud, network intrusions, unusual patterns of disease and ecosystem disturbances, such as droughts, floors, fires, hurricanes, etc.

Often, practices such as preprocessing can impact the effectiveness of an anomaly detection algorithm. For example, in [3] the authors developed early warning systems for flooding

and tested these at the Guillemard Bridge station at the Kelantan River in Malaysia. By performing a preprocessing step using a novel use of algebraic topology (called persistent homology), the number of false alarms reduced from six to four. This was a reduction in rate from 33.33% to 25%.

1.4 Roadmap for the Course

The basic format of the course will be a weekly 1-hour lecture, along with a 2-hour lab session. The main focus of the labs will be to use Python in conjunction with what we see in the lecture, but we will also see SQL. For each lab session you will have a .ipynb file which will guide you through some key features of Python. I would strongly advise you to make purposeful mistakes and make sense of the output, try to make sense of each bit of code and, above all, experiment.

A rough roadmap for the course is as follows:

Road map		
Week	Lecture	Lab
1	Introduction and Overview	
2	Data Storage & Databases	SQL
3	Data Storage & Databases	SQL
4	Types of Data & Data Cleaning	Importing Data and Data Exploration
5	Data Preprocessing	Data Preprocessing
6	Measures of Similarity and Dissimilarity	Data Wrangling
7	Classification	Regression: Lab
8	Classification	Classification: Lab
9	Classification	Classification: Lab
10	Cluster Analysis	Cluster Analysis: Lab
11	Cluster Analysis	Cluster Analysis: Lab
12	Submission of report	

There will be three pieces of assessment. The first is a 'weekly' quiz[1] which is worth 20%. The purpose here is to summarise what has been done in the lectures and get you working on some problems. The second will be a SQL and Database assessment where you will be tasked with creating your own database and using SQL to explore it. This will be worth 25%. Finally, there will be the main coursework project which will be worth the remaining 55%. For this you will be given two topics (one we cover in class, one we do not) in which you must apply both to a dataset(s) and compare and contrast the results. In effect, you will be performing data mining on this dataset(s). This will be worth 50% of the total grade, with the final 5% being a small questionnaire. Here, you will be asked to summarise key aspects of your report.

[1]It may not be *every* week, but it will be most weeks.

1.5 Further Reading

The main lectures will follow the book "Introduction to Data Mining" by Pang-Ning Tan, Michael Steinbach, Anuj Karpatne and Vipin Kumar.

Some other books which may be of use are as follows:

- *Data Mining: Introductory and Advanced Topics*, by M.H. Dunham (Pearson)
- *Data Mining: Concepts and Techniques*, Third Edition, by J. Han, M. Kamber, J. Pei (MK Publishers)
- *Principles of Data Mining*, by D. Hand, H. Mannila, P. Smyth (MIT Press)

The main python sessions will follow the book "Python for Data Analysis: Data Wrangling with Pandas, Numpy and iPython" by Wes McKinney.

Some other books which may be of use are as follows:

- *Python Cookbook*, Third Edition, by David Beazley and Brian K. Jones (O'Reilly)
- *Fluent Python*, by Luciano Ramalho (O'Reilly)
- *Effective Python*, by Brett Slatkin (Pearson)
- *Python Crash Course*, Second Edition, by Eric Matthes (No Starch Press)

Part II

Data Preprocessing

Chapter 2

Data

Further Reading: Chapter 2 of [2], Chapter 2 of [6].

Recall our image of the entire Knowledge Discovery process. The first step is to input data. Really, we should also be concerned about how we select our data but as this is often subject specific, we will mostly assume this is done. As such, we now have a bunch of data and we want to get it in the right form. But to do this, we need to get a grip on what we mean by the right form. At the heart of this is an understanding of the type of data we are dealing with, and the quality of the data.

For the former, are we dealing with qualitative or quantitative data? Does the data have any special characteristics, such as explicit relationships? For the latter, the issue of quality is a direct consequence of how we get our data. Often, data are collected for some purpose other than the data mining analysis itself. For example, it may have been collected in order to maintain an up-to-date record of all the transactions in a bank. This is in stark difference to much of statistics, in which data are often collected by using efficient strategies to answer specific questions. A result of all this is that we may have noise and outliers, as well as missing, inconsistent or duplicate data, not to mention data measured using different units.

In a rough sense, the better quality data we have, the better the quality of the resulting analysis. Improving the data will be discussed soon, but first we need to get a handle on the different types of data we can encounter. Once we do, then perhaps we can gain an understanding of how we want that data to look and behave.

Before we begin, let's consider a rudimentary example. Suppose we want to build a model which would allow us to predict a person's annual credit card spending given their annual income. The selection of data is now done for us. Next, we need to look at how the data was collected. Is it accurate? Do we have any missing entries, such as if our individual was out of the country? Are all the entries in the same currency? Were there any one off payments that may skew the data? These are some of the questions we would need answer in the preprocessing step.

Once complete, we move on and need to decide which model we will use. A simple way to do this is to use regression analysis. In its simplest form, this involves building a predictive model to relate a predictor variable x (in this case, annual income) to a response variable y (in this case, annual credit card spending) through a relationship of the form $y = ax + b$. Clearly this model would not be perfect, but since spending typically increases with income,

the model might be adequate. This is where we need to have some understanding of what needs to be output. If we need more accuracy and detail, this would perhaps be unsuitable. If not, it would be fine. For now, we assume that a rough characterisation is sufficient. In this case, linear regression is a good choice since it is quite simple and unless the data set is very large, few data management problems will be expected. Simple summaries of the data (sums, sums of squares, sums of products etc.) are sufficient to compute estimates of a and b. This means that a single pass through the data will yield results.

Next, we need to decide how to quantify and compare how well our model fits the data. Sometimes this is written as choosing a 'score' function. In this situation, the most commonly used score function is the sum of the squared discrepancies between the predicted spending from the model and the observed spending in the person (or group of people) described by the data. The smaller the sum, the better the model fits the data. These kinds of optimisation problems (which we will see later) are quite simple in the case of linear regression. We would be able to express a, b as explicit functions of the observed values of spending and income.

Finally, we would need to decide what information we output, and what form it takes. For something like linear regression, a good choice might be a simple graph. In general, simplicity of output is preferable to complexity, everything else being equal.

2.1 Types of Data

A *data set* is a set of measurements taken from some environment or process. This can be viewed as a collection of *data objects*[1]. In turn, data objects are typically described by their *attributes*[2]. These capture some characteristic of the object, such as the mass of a physical object, or the time at which an event occurred. If the data objects are stored in a database, they are *data tuples*; that is, the rows of a database correspond to the data objects, and the columns correspond to the attributes. For example, a database could consist of the entire student body of the University of Hertfordshire. In this case, each row corresponds to a student, and each column an attribute, such as Student ID, Year of Study, Course, Modules Taken, Grades, and so on.

With this in mind, let's introduce some definitions.

Definition 2.1. *An attribute is a property or characteristic of an object that can vary, either from one object to another, or from one time to another.*

The names given in the footnote below are often used interchangeably, though there tends to be certain trends within disciplines. For example, dimension is commonly used in data warehousing, feature in machine learning, and variable with statisticians. Attribute is more common with data mining, but we will interchange where the need suits us.

Example 2.2. *The following are examples of attributes:*

- {*Blue, Brown, Green, Hazel*} *is a set of possible eye colours.*

[1] Data objects are also called records, points, vectors, patterns, events, cases, samples, instances, observations, or entities

[2] Attributes are also called variables, characteristics, fields, features or dimensions

- *Temperature is an attribute of an object that varies over time.*
- {*Student ID, Name, Home Address*} *are attributes of a student at UH.*
- *Height, weight and sex are all attributes of a patient at a hospital.*

Definition 2.3. *A measurement scale is a rule (function) that associates a numerical or symbolic value with an attribute.*

A set of attributes used to describe a given object is called an attribute vector, or feature vector.

The distribution of data involving one attribute (or variable) is called univariate. A bivariate distribution involves two attributes, and so on.

It should be clear that the properties of an attribute need not be the same as the properties of the values used to measure it. In other words, the values used to represent an attribute can have properties that are not properties of the attribute itself, and vice versa. We can illustrate this with two examples.

Example 2.4. *(a) Consider the attributes of Employee Age and Employee ID for any given employee. Both of these attributes can be represented by integers (if age is taken in years) and for any set of integers, we can compute the average. While it makes sense to talk of the average age of employees, it does not make sense to talk of the average Employee ID. Instead, for this latter attribute, we are only really interested in whether two given IDs are distinct or not.*

(b) Now consider the attributes of Length of line segments. There are two ways we could do this, we could either let the attributes belong to some discrete set {*short, medium, long*}*, or we could assign numbers. However, with the latter, there are many numbers we could assign. We could assign an actual value of length (in cm, say), or we could assign numbers that only capture the order of these line segments (so 1 for shortest, 2 for next shortest, and so on). Again, for the latter, things like average or sum make little sense. If the line segments are actually something like a plank of wood, then we can also talk about an upper bound on the length, whereas there is no upper bound on the set of real numbers, say.*

The point of the above is that we need to know the type of an attribute because it tells us which properties of the measured values are consistent with the underlying properties of the attribute, and therefore allows us to avoid foolish actions such as computing the average Employee ID. The type of an attribute is determined by the set of possible values the attributes can have. Usually, we classify them into one of four categories: nominal, ordinal, interval and ratio.

To discuss these classifications, we first need to identify the properties of numbers that correspond to the underlying properties of the attribute. The following properties (operations) of numbers are typically used to describe attributes:

1. **Distinctness**, $=$ and $\neq$
2. **Order**, $<$, $\leq$, $>$ and $\geq$

3. **Addition**, + and −
4. **Multiplication**, × and /

The four types mentioned above are defined in the following table, along with statistical operations that are valid for each type. Each attribute type possesses all of the properties and operations of the attribute types above it. Consequently, any property or operation that is valid for nominal, ordinal and interval attributes is also valid for ratio attributes, i.e. the attribute types are cumulative.

	Attribute Type	Description	Examples	Operations
Qualitative	Nominal	The values of a nominal attribute are just different names, i.e. nominal values provide only enough information to distinguish one object from another (=, ≠)	postcodes, employee ID, eye colour, sex	mode, entropy, contingency, correlation, χ^2 test
Qualitative	Ordinal	The values of an ordinal attribute provide enough information to order objects (<, >)	hardness of minerals, {good, bad}, grades, street numbers	median, percentiles, rank correlation, run tests, sign tests
Quantitative	Interval	For interval attributes, the differences between values are meaningful, i.e. a unit of measurement exists (+, −)	calendar dates, temperature in Celsius or Fahrenheit	mean, standard deviation, Pearson's correlation, t and F tests
Quantitative	Ratio	For ratio variables, both differences and ratios are meaningful (×, /)	temperature in Kelvin, monetary quantities, counts, age, mass, length, electrical current	geometric mean, harmonic mean, percent variation

Nominal (meaning "relating to names") attributes are symbols or *names of things*. Each value represents some kind of category, code or state, and so nominal attributes are also referred to as categorical. They do not have any meaningful order and things like mean (average) or median (middle) have no meaning here. One thing that may be of interest, however, is the attributes most commonly occurring value. This value, knows as the mode, is one of the measures of central tendency.

Ordinal attributes are differentiated from nominal values by the fact that ordering is now meaningful. In this way, we can rank ordinal values. For example, *drink_size* could correspond to the size of drinks in a fast-food restaurant. It could have three possible values: *small, medium* and *large*. These values have a meaningful sequence corresponding

to increasing drink size, and so are ordinal. Another example could be grades, military rank or certain job titles. Ordinal attributes are useful for registering subjective assessments of qualities that cannot be measured objectively. They are therefore often used in surveys for ratings (e.g. a survey on customer satisfaction). Note that qualitative attributes can be assigned numeric values without becoming quantitative. We have already seen the example of employee/student ID, but you could also assign numbers to categories in the aforementioned customer satisfaction survey, as follows: 0 - *very dissatisfied*, 1 - *somewhat dissatisfied*, 2 - *neutral*, 3 - *somewhat satisfied*, 4 - *very satisfied*. The central tendency of an ordinal attribute can be represented by its mode and its median (the middle value in an ordered sequence), but the mean cannot be defined.

Interval-scaled attributes are measured on a scale of equal-size units. The values of interval-scaled attributes have order and can be positive, zero or negative. Thus, not only can we provide a ranking of values, but we can also compare and quantify the difference between values. This makes them quantitative, rather than qualitative. They are measurable quantities, presented as integer, rational or real values.

A typical example is temperature. If we measure the temperature outside our window each day (the day, here, is the object) for a month, then we can order these values, obtain a ranking of which days were hottest/coldest, and quantify the swing in temperature(s). There is a subtlety here in how we choose to measure temperature. When measured on the Kelvin scale, a temperature of $2°$ is, in a physically meaningful way, twice that of a temperature of $1°$. This is not the case when temperature is measured in Fahrenheit or Celsius where, physically, a temperature of $1°$ Fahrenheit (Celsius) is not much different than a temperature of $2°$ Fahrenheit (Celsius). The reason for this is the somewhat arbitrary zero point for Fahrenheit and Celsius, i.e. $0°$ (Fahrenheit or Celsius) do not indicate 'no temperature'. On the Celsius scale, for example, the unit of measurement is $\frac{1}{100}$ of the difference between the melting temperature and the boiling temperature of water in atmospheric pressure. As such, for Fahrenheit and Celsius we can compute the *difference* between temperature values, but we cannot talk of one temperature value as being a multiple of another. For this reason, it is interval-scaled, and not ratio-scaled. Another example is that of calendar dates. The year 0 does not correspond to the beginning of time. It is an arbitrary choice and so is not a ratio-scaled attribute. For interval-scaled attributes, we can compute their mean value, median and mode.

Finally, a ratio-scaled numeric attribute is differentiated from an interval-scaled numeric attribute by the presence of an inherent zero-point. This allows us to make statements such as 'Attribute A is twice that of Attribute B'. So these values are ordered and we can compute the difference between values, multiples of values, as well as the mean, median and mode.

As we saw above, temperature measured in Kelvin is ratio-scaled as $0°$ $(= -273°C)$ is a true zero-point. It is the point at which the particles that comprise matter have zero kinetic energy. Another example is that of *years_of_experience* or *number_of_words*. We could also include things like height, weight etc.

We can also describe the above attributes in terms of transformations that do not change the meaning of an attribute. This was the way in which S. Smith, the psychologist who defined these types of attributes, originally defined them. For example, if we measure length in terms of metres or feet, it does not change the meaning of the length attribute.

	Attribute Type	Transformation	Comment
Qualitative	Nominal	Any one-to-one mapping. These are also called injective mappings. An example would be any permutation of values.	If all employee ID numbers are reassigned, it will not make any difference.
Qualitative	Ordinal	An order-preserving change of values, i.e. for some monotonic f, $new\_value = f(old\_value)$.	An attribute encompassing the notion of good, bad can be represented equally well by the values $\{0, 1\}$ or $\{2, 4.5\}$. You could relabel flat letters $A, B, C, \ldots$ with 1, 2, 3, ...
Quantitative	Interval	$new\_value = a \times old\_value + b$, where a, b constants	The Fahrenheit and Celsius temperature scaled differ in the location of their zero value and the size of a degree (unit).
Quantitative	Ratio	$new\_value = a \times old\_value$, where once again a is a constant	Length can be measured in meters or feet. Kelvin has a well-defined zero-point.

So far, we have organised attributes into nominal, ordinal, interval and ratio types. There are many ways to organise attribute types and the types are not mutually exclusive. Classification algorithms developed from the field of machine learning often talk of attributes as being either *discrete* or *continuous*. Each type may be processed differently.

A *discrete* attribute has a finite or countably infinite set of values. Such attributes can be categorical, such as ID numbers, or numeric such as counts. Discrete attributes are often represented using integer variables, but do not have to. A special case of a discrete attribute is binary. Here, we assume only two values (usually true/false or 1/0) and are often represented as Boolean variables. More generally, the attributes *hair_colour, smoker, medical_test* and *drink_size* each have a finite number of values, and so are discrete. We should contrast these with something like *student_ID* for which the possible values are infinite, but can be put into a one-to-one correspondence with the natural numbers. Such an attribute is called *countably infinite.*

If an attribute is not discrete, it is *continuous.* Here, we are really thinking of the real numbers. Examples include attributes such as *temperature, height* and *weight.* Continuous attributes are typically represented as floating-point variables. If we are being pedantic, we never really work with the real numbers, but with approximations of them. In practice, we can only with with limited precision and real numbers are represented using a finite number of digits.

We also introduce the idea of symmetry and asymmetry. For asymmetric attributes, only presence (a non-zero attribute value) is regarded as important. Consider a data set in which each object is a student and each attribute records whether a student took a particular course at a university. For a specific student, an attribute has a value of 1 if the student took the course associated with that attribute, and a value of 0 otherwise. Because students take

only a small fraction of all available courses, most of the values in such a data set would be 0. Therefore, it is more meaningful and more efficient to focus on the non-zero values. Binary attributes in which the outcomes are not equally important are called *asymmetric* (e.g. a medical test where the only outcome is positive and negative). By convention, we code the most important outcome, which is usually the rarest, by 1 (e.g. HIV *positive* denoted by 1, and HIV *negative* denoted by 0). A binary attribute in which both of its states are equally valuable and carry the same weight (e.g. *sex* having the states *male/female*) are called *symmetric*. It is also possible to have more general discrete (or even continuous) asymmetric features, e.g. the number of credits associated with each module is recorded and will be asymmetric.

With an understanding of attributes, we now discuss the types of data set we can encounter. We group these in three types: record data, graph-based data and ordered data. Note that these do not cover all possibilities, but will cover the main kinds of data.

Before discussing the above, we introduce three characteristics that will apply to many data sets and have a significant impact on the data mining techniques that are used.

- **Dimensionality**

  This is the number of attributes that the objects in the data set possess. Analysing data with a lower number of dimensions tends to be qualitatively different from analysing moderate- or high-dimensional data. The difficulties associated with high-dimensional data is sometimes referred to as the *curse of dimensionality*. For this reason, dimensionality reduction algorithms are well-studied and sought after. This is part of data preprocessing and can include, among other things, principal component analysis.

- **Distribution**

  This is the frequency of occurrence of various values or sets of values for the attributes comprising data objects. It can be considered as a description of the concentration of objects in various regions of the data space. There are many types of distribution (e.g. Gaussian (normal)) and statistical approaches arising from these can yield powerful analysis techniques.

  Despite the above, many data sets have distributions that are not well captured by standard statistical distributions. For this reason, many data mining algorithms do not assume a particular statistical distribution. Situations may arise where, for example, one category may arise 95% of the time, while all other categories arise just 5% of the time. This *skewness* in the distribution can make classification difficult. A special case of this is *sparsity*. A typical example is that of students and the courses/modules they are sitting. In this case we have binary data (0 for not taking a given course/module, 1 for taking it) in which fewer than 1% (say) of the values are non-zero. In this case, sparsity is an advantage because we are only interested in non-zero values. We would therefore have significant gains with respect to computation time and storage.

- **Resolution**

  It is frequently possible to obtain data at different levels of resolution, and often the properties of the data are different at different resolutions. For example, the surface of

the Earth seems very uneven at a resolution of a few meters, but is relatively smooth at a resolution of tens of kilometres. The patterns we are looking for, then, are dependent on the resolution chosen. Another example involves variations in atmospheric pressure. On a scale of hours this may reflect the movement of storms and other weather systems, but on the scale of months such phenomena are not detectable.

With the above in mind, we now discuss each of record data, graph-based data and ordered data.

- Record Data

A lot of data mining assumes that the data set is a collection of records (data objects), each of which consists of a fixed set of data fields (attributes). This could include data objects which are individuals, and whose attributes include annual income, marital status, employment status etc. For the most basic form of record data, there is no explicit relationship among records or data fields, and every record (object) has the same set of attributes. Record data is usually stored either in flat files or in relational databases.

Another kind of record data includes *transaction, or market based data*. In this case, each record (transaction) involves a set of items. A typical example is a shop in which the set of products purchased by a customer during one shopping trip constitutes a transaction, while the individual products that were purchased are the items. Most often, the attributes are binary, indicating whether an item was purchased, but more generally the attributes can be discrete or continuous, representing the number of items purchased or the amount spent on those items.

If all the data objects in a collection of data have the same fixed set of numeric attributes, then the data objects can be thought of as points (vectors) in a multidimensional space, where each dimension represents a distinct attribute describing the object. This set can then be represented as an $m \times n$ matrix, where each row denotes an object, and each column denotes an attribute. This is called a *data matrix*, or sometimes a *pattern matrix.*
e.g.

Projection of x load	Projection of y load	Distance	Load	Thickness
10.23	5.27	15.22	27	1.2
12.65	6.25	16.22	22	1.1
13.54	6.25	16.22	23	1.2
14.27	8.43	18.45	25	0.9

This can be represented as a matrix

$$X = \begin{pmatrix} 10.23 & 5.27 & 15.22 & 27 & 1.2 \\ 12.65 & 6.25 & 16.22 & 22 & 1.1 \\ 13.54 & 6.25 & 16.22 & 23 & 1.2 \\ 14.27 & 8.43 & 18.45 & 25 & 0.9 \end{pmatrix}.$$

As a data matrix consists of numeric attributes, standard matrix operations can be applied to transform and manipulate the data. Therefore, the data matrix is the standard data format for most statistical data.

- Graph-based Data

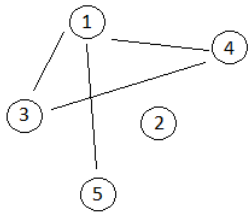
We consider two specific cases: (1) the graph captures relationships among data objects, and (2) the data objects themselves are represented as graphs.

Considering (1) first, the relationships among objects frequently convey important information. To reflect this, we often represent the data as a graph. Here, we mean graph in the mathematical sense, i.e. a graph is a pair (V, E), where V is a set of *nodes*, and E is a set of *edges* between the nodes. Here, we may think of $E \subset V \times V$ as being a subset of $V \times V$ where the first entry represents the starting node, and the second entry denoting the end node. Graphs can be directed (i.e. the direction you travel along the edge matters) or undirected (i.e. the starting node and ending node do not really matter).

Example 2.5. *Suppose the node set is* $V = \{1, 2, 3, 4, 5\}$. *Then we can define a graph on this set with edge set given by,*

$$E = \{(1, 3), (1, 4), (1, 5), (3, 1), (3, 4), (4, 1), (4, 3), (5, 1)\}.$$

Notice that no edges go to, or come from, 2, and this is reflected in E. *We would say this graph is disconnected since it has one connected chunk involving* 1, 3, 4, 5, *and another connected chunk involving just 2. Notice that for each edge, it does not matter which end we start at. As such, we say it is an undirected graph. We can visualise this as follows:*



In our case, we can represent data sets as graphs (in the above sense) by mapping objects to nodes, and relationships between objects to edges. A typical example might be web pages on the internet, where each web page is a node, and each edge is a link from one web page to another. A similar example might be Twitter where each account is a node, and an edge is formed from A to B when A follows B. This would give us a directed graph since, when A follows B it is not a requirement that B also follows A. This is a fascinating area that has seen several ingenious strategies employed to explore relationships. A very modern approach uses topological structures such as sheaves (cf. [5] and `https://thepenngazette.com/%EF%BB%BFflipping-the-script/`), but that is far outside the scope of this course.

For (2), if objects have structure (i.e. the objects contain subobjects that have relationships), then such objects are frequently represented by a graph. For example, the structure of chemical compounds can be represented by a graph, where the nodes are atoms and the links between nodes are chemical bonds (see Figure 2.1 where the benzene molecule is represented with atoms of carbon (black) and hydrogen (grey)). A graph representation makes it possible to determine which substructures occur frequently in a set of compounds and to ascertain whether the presence of any of these substructures is associated with the presence or absence of certain chemical properties, such as melting point or heat of formation.

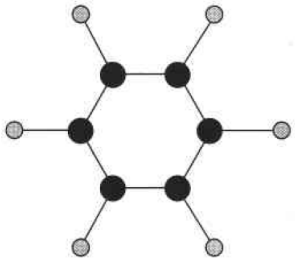


Figure 2.1: Benzene molecule. [2]

Time	Customer	Items Purchased
t1	C1	A, B
t2	C3	A, C
t2	C1	C, D
t3	C2	A, D
t4	C2	E
t5	C1	A, E

Customer	Time and Items Purchased
C1	(t1: A,B) (t2:C,D) (t5:A,E)
C2	(t3: A, D) (t4: E)
C3	(t2: A, C)

(a) Sequential transaction data.

```
GGTTCCGCCTTCAGCCCCGCGCC
CGCAGGGCCCGCCCCGCGCCGTC
GAGAAGGGCCCGCCTGGCGGGCG
GGGGGAGGCGGGGCCGCCCGAGC
CCAACCGAGTCCGACCAGGTGCC
CCCTCTGCTCGGCCTAGACCTGA
GCTCATTAGGCGGCAGCGGACAG
GCCAAGTAGAACACGCGAAGCGC
TGGGCTGCCTGCTGCGACCAGGG
```

(b) Genomic sequence data.

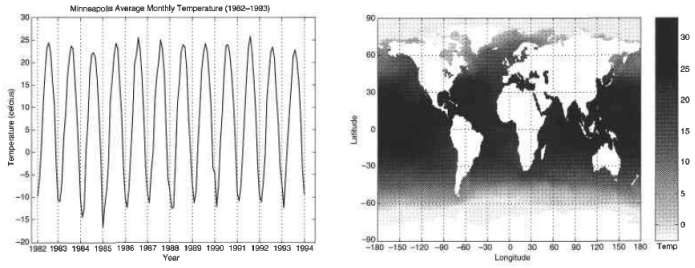


Figure 2.2: Different variations of ordered data. [2]

- Ordered data

For some types of data, the attributes have relationships that involve order in time or space. One natural way to encounter this is with *sequential transaction data*. This is a natural extension of transaction data, where now each transaction has a time associated with it. As such, you can not only view spending data, but can track spending patterns over time. In this way, one can identify spending patterns such as around holidays, or view spikes in certain items following newsworthy events. In Figure 2.2(a), we see an example with five different times (t_1, t_2, t_3, t_4, t_5), three different customers (C_1, C_2, C_3) and five different items (A, B, C, D, E). In the top table, each row corresponds to the items purchased at a particular time by each customer. In the bottom table, the same information is displayed, but each row corresponds to a particular customer. Each row contains information about each transaction involving the customer, where a transaction is considered to be a set of items and the time at which those items were purchased.

Another example of ordered data is *time series data.* This time, each record is a time series (i.e. a series of measurements taken over time). A typical example might be a financial data set containing objects that are daily prices of various stocks. Another example (see Figure 2.2(c)) could be average monthly temperature of Minneapolis during the years 1982 to 1994. When working with temporal data, such as time series, it is important to consider *temporal autocorrelation*, i.e. if two measurements are close in time, then the values of those measurements are often very similar.

A third example is *sequence data* which consists of a data set that is a sequence of individual entities, such as words or letters. It is similar to sequential data, except there are no time stamps; instead, position is important. For instance, the genetic information of plants and animals can be represented in the form of sequences of genes. Many of the problems associated with genetic sequence data involve predicting similarities in the structure and function of genes from similarities in the nucleotide sequences. See Figure 2.2(b) which shows a section of the human genetic code.

Finally, *spatial and spatio-temporal data* is an example of ordered data. Some objects have spatial attributes, such as position or area, in addition to other types of attributes. A standard example of this is weather data (precipitation, temperature, pressure) that is collected at several geographical locations. Often, such measurements are collected over time, thereby providing an evolving picture from which patterns can be determined. Although analysis can be conducted separately for each specific time or location, a more complete analysis of spatio-temporal data requires consideration of both the spatial and temporal aspects of the data. Important examples of spatio-temporal data are science and engineering data sets such as Earth science data which records the temperature and/or pressure at points on latitude-longitude spherical grids of various resolutions (e.g. 1° by 1°). See Figure 2.2(d).

As with time series data, an important aspect of spatial data is *spatial autocorrelation*, i.e. objects that are physically close tend to be similar in other ways as well. Thus, two points on Earth that are close physically are likely to have similar values for temperature and rainfall.

2.2 Summary Statistics

Before, and even during, the preprocessing stage, it is essential to have an overall picture of our data. Basic statistical descriptions can be used to identify properties of the data and highlight which data values should be treated as noise or outliers. In the interests of keeping things brief, we will focus on three areas of basic statistical descriptions: measures of central tendency; dispersion of the data; graphic displays of basic statistical descriptions to visually inspect our data.

Roughly speaking, the first of these three measures the location of the middle/centre of our data. Intuitively, we are asking where most of an attribute's values fall. Related to this will be the mean, median, mode and midrange. The second of these assesses the central tendency of our data set. In other words, we want to know how spread out our data is. Here, we are interested in variance, standard deviation, range, quartiles and interquartile ranges. Finally, we want to visually inspect our data, and so use bar charts, pie charts and line graphs, among other displays.

2.2.1 Measuring Central Tendency

Suppose we have some attribute X (could be salary, height etc.) which has been recorded for a set of objects. We let $\{x_1, x_2, \ldots, x_N\}$ be the set of N observations for X. We can also refer to these values as the data set for X. If we were to plot the observations for salary/height, where would most of the values fall? This gives us an idea of the central tendency of the data. However, this is not a very rigorous definition and measures of central tendency include the mean, median, mode and midrange.

- Mean

The most common numeric measure of the 'centre' of a set of data is the (arithmetic) mean. For the data set above, the mean of this set of values is given by,

$$\bar{X} = \frac{1}{N}\sum_{i=1}^{N} x_i.$$

e.g. Suppose the following values (in thousands) are the salaries of 11 individuals, shown in increasing order: 17, 20, 23, 31, 40, 55, 60, 62, 64, 84, 99. Then, $\bar{x} = \frac{1}{11}(17+20+\cdots+99) = 50.45k$.

Sometimes, each value x_i in a set may be associated with a weight w_i for $i \in \{1, \ldots, N\}$. The weight reflects the important, or occurrence frequency attached to their respective values. In this case, we can compute the *weighted arithmetic mean/weighted average* as follows:

$$\bar{x}_W = \frac{\sum_{i=1}^{N} w_i x_i}{\sum_{i=1}^{N} w_i}.$$

The benefit of the mean (and weighted mean) is that is is easy to compute and gives a useful insight into the data set. However, it does have a major problem since it is sensitive to extremes (e.g. outliers). For example, if in the above example we included one more individual, whose salary was £26,800,000 (looking at you, Cristiano Ronaldo) then the average will be massively skewed, even though it is just one person. A classic case of this is when discussing life expectancy of earlier periods in history. For example, in Victorian times the average age for a baby boy was just 40. However, this is in large part a consequence of the high infant mortality rate in such times. Once this is stripped out, life expectancy at 5 years was 75 for men, remarkably similar to today. Indeed, life expectancy for mature men has not changed dramatically for over 3000 years (the same is not true, however, for women). See `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2625386/pdf/577.pdf` for a snapshot of this.

To offset the effect caused by a small number of extreme values, we can instead use the *trimmed mean*, which is the mean obtained after removing a few values at the high and low extremes. In this case, a percentage p between 0 and 100 is specified, the top and bottom $(\frac{p}{2})\%$ of the data is thrown out, and the mean is then calculated in the normal way. In this way, we can think of it as a generalisation of the mean where the standard mean corresponds to $p = 0\%$.

e.g. Consider the values {1, 2, 3, 4, 100}. Then the mean of these values is $\frac{110}{6} \approx 18.3$. However, if we take the trimmed mean with $p = 40\%$ then the trimmed mean is 3.

Of course, the key is to trim off the right amount. Too much and we may lose valuable information. Too little and we aren't removing the problem.

- Median

For skewed (asymmetric) data, a better measure of the centre of data is the *median*, which is the middle value in a set of ordered data values. It is the value that separates the higher half of a data set from the lower half.

To compute, let $X = \{x_1, \ldots, x_N\}$ be sorted in non-decreasing order[3] so that $x_1 = min(X)$ and $x_N = max(X)$. Then the median is defined as follows:

$$median(X) = \begin{cases} x_{r+1}, & \text{if } m \text{ is odd, i.e. } m = 2r + 1; \\ \frac{1}{2}(x_r + x_{r+1}), & \text{if } m \text{ is even, i.e.} m = 2r. \end{cases}$$

So, we organise our data set so the the values are increasing. If the cardinality of our set is odd, we take the value that is bang in the middle, splitting the set into two equal parts. You can think of this as proceeding one step at a time where each step involving throwing away the smallest and largest element. You keep going until you end up in the middle. If the cardinality is even, then you do exactly the same, but now you are left with two elements. So all you do is take the mean of them.

e.g. Go back to {1, 2, 3, 4, 100}. As it is ordered, the first step is to remove 1 and 100. Then we remove 2 and 4. We are therefore left with 3 and so that is the median.

Suppose we add an element so our set becomes {1, 2, 3, 4, 5, 100}. Well we do the same thing but are now left with {3, 4}. Take the mean of these and so the median in this case is 3.5.

Observe that the median is just the trimmed mean with $p = 100\%$.

Example 2.6. *Recall, in the last chapter we encountered the Iris data set. From this, we can obtain the following percentiles for the four quantitative attributes of this data set.*

[3]There are many popular sorting algorithms such as merge sort, quicksort, bubble sort, insertion sort. Investigating these is left as an exercise.

All values in cm				
Percentile	*Sepal Length*	*Sepal Width*	*Petal Length*	*Petal Width*
0	*4.3*	*2.0*	*1.0*	*0.1*
10	*4.8*	*2.5*	*1.4*	*0.2*
20	*5.0*	*2.7*	*1.5*	*0.2*
30	*5.2*	*2.8*	*1.7*	*0.4*
40	*5.6*	*3.0*	*3.9*	*1.2*
50	*5.8*	*3.0*	*4.4*	*1.3*
60	*6.1*	*3.1*	*4.6*	*1.5*
70	*6.3*	*3.2*	*5.0*	*1.8*
80	*6.6*	*3.4*	*5.4*	*1.9*
90	*6.9*	*3.6*	*5.8*	*2.2*
100	*7.9*	*4.4*	*6.9*	*2.5*

From this we can compute the means, medians and trimmed means ($p = 20\%$) of these four quantitative attributes. Observe the three measures of central tendency have similar values except for the attribute petal length.

All values in cm				
Measure	*Sepal Length*	*Sepal Width*	*Petal Length*	*Petal Width*
mean	*5.84*	*3.05*	*3.76*	*1.20*
median	*5.80*	*3.00*	*4.35*	*1.30*
trimmed mean (20%)	*5.79*	*3.02*	*3.72*	*1.12*

While the median benefits from being easy to compute, it is expensive to compute when we have a large number of observations. Nevertheless, for numeric attributes we can easily *approximate* the value. Assume that the data are grouped in intervals according to their x_i data values and that the frequency (i.e. the number of data values) of each interval is known. For example, we can group employee's annual salaries into the intervals £10,000-£20,000, £20,000-£30,000, and so on. Let the interval that contains the median frequency be the *median interval*. We can then approximate the median of the entire data set (e.g. the median salary) by interpolation using the formula,

$$median \approx L_1 + \left(\frac{\frac{N}{2} - (\sum freq)_l}{freq_{median}} \right) width,$$

where

- L_1 is the lower boundary of the median interval;
- N is the number of values in the entire data set;
- $(\sum freq)_l$ is the sum of the frequencies of all of the intervals that are lower than the median interval;
- $freq_{median}$ is the frequency of the median interval;

- *width* is the width of the median interval.

- Mode

The mode is another measure of central tendency. For a set of data, the mode is the value that occurs most frequently in the set. Therefore, it can be determined for qualitative and quantitative attributes. It is possible for the greatest frequency to correspond to several different values, which results in more than one mode. Data sets with one/two/three modes are respectively called *unimodal/bimodal/trimodal*. In general, a data set with two or more modes is *multimodal*. At the other extreme, if each data value occurs only once, then there is no mode.

e.g. The mode of $X = \{1, 1, 2, 3, 4, 4, 4, 5, 6, 7\}$ is 4 (unimodal) and the mode of $Y = \{1, 1, 2, 2, 3, 3, 3, 4, 4, 4, 5, 6, 6, 7\}$ is 3 and 4 (bimodal).

For unimodal numeric data that are moderately skewed (asymmetrical), we have the following approximation:

$$mean - mode \approx 3 \times (mean - median).$$

Thus, we can approximate the mode so long as we know the mean and median.

- Midrange

The *midrange* is the average of the largest and smallest values in the set. This measure is easy to compute using the SQL aggregate functions `max()` and `min()`.

e.g. Return to the salary example from earlier, with values (in thousands) given by 17, 20, 23, 31, 40, 55, 60, 62, 64, 84, 99. Then the midrange is $\frac{17000+99000}{2} = 58000$.

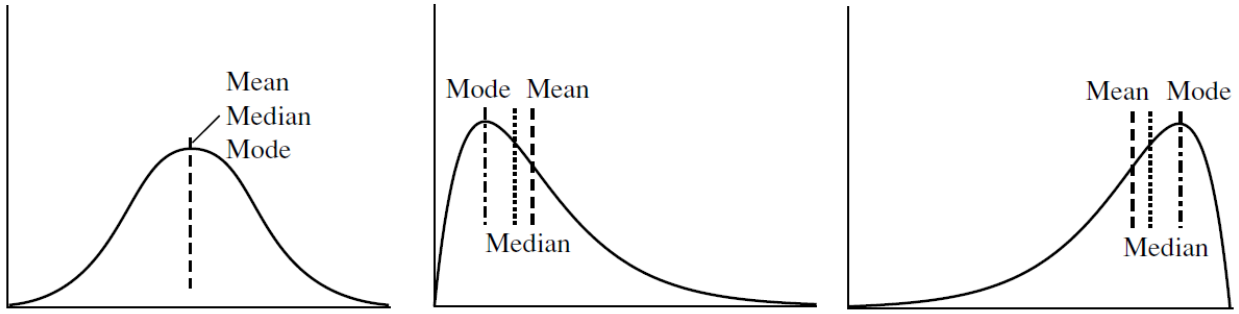
We can conclude by noting that in a unimodal frequency curve with perfect symmetric data distribution, the mean, median and mode are all the same center value, as shown in Figure 2.3(a). Of course, data in most real applications are not symmetric. Instead, they may either be *positively skewed* (where the mode occurs at a value that is smaller than the median) as seen in Figure 2.3(b), or *negatively skewed* (where the mode occurs at a value greater than the median) as seen in Figure 2.3(c).

Importantly, the above can be generalised for measures of location for data that consists of several attributes (multivariable/multivariate data), $\boldsymbol{x} = (x_1, \ldots, x_N)$. For computing the mean and median, the we just do the obvious; namely, we compute the mean and median for each attribute separately. So, if $\bar{x_i}$ is the mean of the i^{th} attribute, then

$$\bar{\boldsymbol{x}} = (\bar{x_1}, \ldots, \bar{x_N}).$$

2.2.2 Measuring the Dispersion

In the interests of completeness, we first outline *range, quantiles, quartiles, percentiles* and the *interquartile range*. Again, let $x_1, \ldots, x_N$ be a set of observations for some numeric attribute X. The *range* of the set is the difference between the largest and the smallest values. If these are ordered so that $x_1 = min$ and $x_N = max$, then this is simply $x_N - x_1$.



(a) Symmetric data **(b)** Positively skewed data **(c)** Negatively skewed data

Figure 2.3: Mean, median and mode of symmetric versus positively/negatively skewed data [6]

Keep this assumption that our data is ordered. Imagine that we can pick certain data points so as to split the data distribution into equal-size consecutive sets. For example, if our set is $\{x_1, x_2, x_3, x_4, x_5, x_6\}$ then we might split this set into $\{x_1, x_2\} \cup \{x_3, x_4\} \cup \{x_5, x_6\}$. These data points are called *quantiles* and they are points taken at regular intervals of a data distribution, dividing it into essentially equal size sets. The reason for 'essentially' here is because our set may not equally divisible, for instance if the above set had just five points instead of six. The k^{th} *q-quantile* for a given data distribution is said to be the value x such that, at most, $\frac{k}{q}$ of the data values are less than x, and at most $\frac{q-k}{q}$ of the data values are more than x, where k is an integer such that $0 < k < q$. There are $q - 1$ q-quantiles.

The 2-quantile is the data point dividing the lower and upper halves of the data distribution. It corresponds to the median. The 4-quantiles are the three data points that split the data distribution into four equal parts (each part represents one-fourth of the data distribution). They are more commonly referred to as quartiles. The 100-quantiles are more commonly referred to as percentiles, which we have already met in the Iris data set. They divide the data distribution into 100 equal-sized consecutive sets. The median, quartiles and percentiles are the mostly widely used forms of quantiles.

The quartiles give an indication of a distribution's centre, spread and shape. The first quartile (denoted by Q_1) is the 25^{th} percentile. It cuts off the lowest 25% of the data. The second quartile is the 50^{th} percentile. As the median, it gives the centre of the data distribution. The third quartile, denoted by Q_3 is the 75^{th} percentile. It cuts off the lowest 75% of the data. The distance between the first and third quartiles is a simple measure of spread that gives the range covered by the middle half of the data. This distance is called the *interquartile range (IQR)* and is defined as

$$IQR = Q_3 - Q_1.$$

e.g. Suppose we have a set $\{1, 3, 4, 4, 5, 6, 8, 14, 14, 20, 22, 22\}$ which has 12 data points. The quartiles are the three values that split the sorted data into four equal parts. So, in this case, the quartiles are the third, sixth and ninth values. Thus, $Q_1 = 4$, $Q_2 = 6$, $Q_3 = 14$ and $IQR = 14 - 4 = 10$.

We now conclude this subsection with a discussion of variance and standard deviation. These indicate how spread out a data distribution is. A low standard deviation means that the data tends to be very close to the mean, while a high standard deviation indicates that the data are spread out over a large range of values. As usual, we let $x_1, \ldots, x_N$ be our N observations of a numeric attribute X. Then the variance σ_X^2 of X is given by,

$$\sigma_X^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2.$$

The thinking here is that we want to know how far away each data point is from the mean. We then want to sum up these differences to get a picture of the total variance from the mean. The only problem is that sum values are less than the mean (so will give a negative value), while some are greater than the mean (so will give a positive value). Thus, we will get cancellations. In fact, we will get 0, as the following shows:

$$\begin{aligned}\sum_{i=1}^{N}(x_i - \bar{x}) = &\quad \textstyle\sum_{i=1}^{N}(x_i) - N\bar{x} \\ = &\quad \textstyle\sum_{i=1}^{N} x_i - \sum_{i=1}^{N} x_i \\ = &\quad 0.\end{aligned}$$

This is certainly not what we want, and the problem is coming from the positives and negatives cancelling out. One way around this is to take the absolute value (i.e. just take the number and drop the sign) but this isn't that nice (i.e. it isn't smooth). However, there is another operation which *is* smooth and also kills minus signs - squaring. This is where the variance comes into play and it turns out to have a number of nice properties. One nice property is the presence of the central limit theorem, which is at work whenever we measure the mean and standard deviation of a distribution we assume to be normal (which is a lot, particularly in nature). We can use this theorem to make all sorts of predictions about the entire distribution since a normal distribution is completely specified by its mean and standard deviation.

With the above in mind, we have introduced a square which in some ways is not what we want. So, to undo this in some sense, we can take the square root. This is what we call the standard deviation. In other words, the standard deviation is given by

$$\sigma_X = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}.$$

At this point, it is worth mentioning a subtlety which can cause problems. In the literature, you may see a definition for variance that has denominator N, *not* $N-1$. The reason for this stems from the idea of bias. Without going into too much detail, if our statistic is *not* an under- or overestimate of a population parameter, then that statistic is said to be unbiased. So, in other words, an unbiased estimator is an accurate statistic that is used to approximate a population parameter. If an over- or underestimate does happen, the mean of the difference is called a bias. As an example, suppose we want to find the average amount

people spend on food per week in the UK. Obviously we cannot survey the whole population, so instead we take a sample of, say, 1,000. From this, we find that the average amount spent on food per week is £60 per person. If £60 is indeed the average for the whole population, then this is unbiased. Otherwise, it is biased. The causes of bias are all to do with how we took our sample. We need to know if the questions were biased (e.g. do we mean how much a person spent in that *given* week, or on an *average* week?) or whether the way we chose the sample was biased or if we have excluded parts of the population, and so on. Our formula (the one with $N-1$ in the denominator) is called an *unbiased* estimate, or the *sample variance*. The use of $N-1$ is called Bessel's correction as it corrects the bias in the estimate of the population variance. It also partially corrects the bias in the estimation of the population standard deviation. On average, with this formula, the sample variance is equal to the unknown population variance (the variance we would get if we did ask every person in the UK how much they spend on food). If there was an N in the denominator, the corresponding formula is called a biased estimate, or the population variance if we did know the actual mean of the whole population (i.e. we did ask everyone in the UK).

In reality, it all boils down to a trade off between bias and variance. If you use the biased version then you'll get less variance but with bias, whereas if you use the unbiased version then you have no bias but more variance. Which you choose is somewhat a matter of preference (within reason). However, in [2] the unbiased version is used and so that is what we will stick with.

Example 2.7. *This example was taken from*

`https://towardsdatascience.com/variance-sample-vs-population-3ddbd29e498a`

and also includes a brief discussion of the above, along with the Bessel factor. I'd recommend you check it out.

So, imagine a forest of 10,000 oak trees. This is the entire population and we want to estimate the distribution of heights. If we counted all 10,00, then we would find out that the heights are normally distributed with an average of 10m and a standard deviation of 2m. These are the statistical parameters of the entire population.

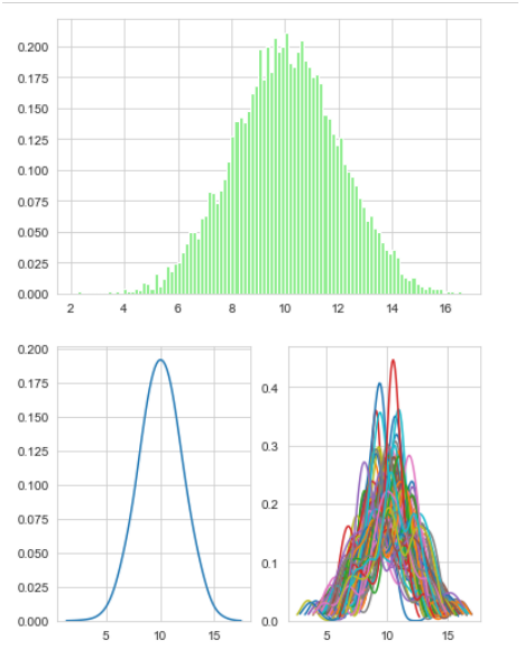
However, we do not want to count 10,000 oak trees. We want to try and estimate these values. So instead, we take a sample of 20 random oak trees, measure their heights and then repeat this experiment 100 times. The following is the code in Python:

```
import numpy as np
import pandas as pd
from random import sample,choice
import matplotlib.pyplot as plt
import seaborn as sns
import scipy

np.random.seed(1234)
mu = 10
sigma = 2
pop = np.random.normal(mu, sigma, 10000)
count, bins, ignored = plt.hist(pop, 100, density=True, color =
'lightgreen')
sns.set_style('whitegrid')
tests = 100
sam = []
mean =[]
std_b = []
std_u = []
fig, axs = plt.subplots(ncols=2)
sns.kdeplot(pop, bw=0.3, ax=axs[0])
for i in range(tests):
    sam_20 = np.random.choice(pop, 20)
    sns.kdeplot(sam_20, bw=0.3, ax=axs[1])
    sam.append(sam_20)
    mean.append(np.mean(sam_20))
    std_b.append(np.std(sam_20))
    std_u.append(np.std(sam_20, ddof=1))

frame = {'mean':mean, 'std_b': std_b, 'std_u': std_u}
table = pd.DataFrame(frame)
```

In this way, for each of the 100 experiments, we get an unbiased ($ddof = 1$ is the same as $N - 1$ instead of N) standard deviation 'std_u', a biased standard deviation 'std_b and a sample mean 'mean'. Graphically, this looks like the following:



Of course, 20 samples are a tiny subset of the 10,000 in the whole population. As such, every time we run the test, we get different distributions. The point is, that on average, we get a reasonable estimate of the real mean and real standard deviation. We can see this in Python using the command `table.describe` *which shows that the unbiased sample standard deviation is on average nearer to the value of the population parameter than the biased one.*

	mean	std_b	std_u
count	100.000000	100.000000	100.000000
mean	10.095473	1.916363	1.966147
std	0.473076	0.279373	0.286630
min	9.104423	1.321537	1.355868
25%	9.781574	1.752017	1.797532
50%	10.046942	1.910105	1.959726
75%	10.479769	2.091813	2.146155
max	11.436420	2.819455	2.892699

Recall, we discussed the sensitivity of the mean to outliers. Since the variance is computed using the mean, it is also sensitive to outliers. Indeed, as we have used the square difference, the variance is particularly sensitive to outliers. As a result, we sometimes want to use more robust estimates of the spread of a set of values. Three such measures are the *absolute average deviation (AAD)*, the *median absolute deviation (MAD)* and the *interquartile range (IQR)* which we have already discussed. The first two are defined as follows:

$$\begin{aligned} AAD = & \quad \tfrac{1}{N}\sum_{i=1}^{N}|x_i - \bar{x}| \\ MAD = & \quad median(\{|x_1 - \bar{x}|, \ldots, |x_N - \bar{x}|\}). \end{aligned}$$

Returning to the Iris data set from earlier, we can compute these measures and compare. This is shown in the following table:

All values in cm				
Measure	Sepal Length	Sepal Width	Petal Length	Petal Width
range	3.6	2.4	5.9	2.4
std	0.8	0.4	1.8	0.8
AAD	0.7	0.3	1.6	0.6
MAD	0.7	0.3	1.2	0.7
IQR	1.3	0.5	3.5	1.5

As with the mean and median, we can generalise measures of dispersion to multivariable data. Once again, we can do this by looking at the spread of each attribute separately. However, for data with continuous variables, the spread of the data is most commonly captured by the *covariance matrix*, S, whose ij^{th} entry s_{ij} is the covariance of the i^{th} and j^{th} attributes of the data. We denote these attributes by x_i, x_j, respectively. Given this, the covariance is given as follows:

$$\begin{aligned} s_{ij} = & \quad covariance(x_i, x_j) \\ = & \quad \tfrac{1}{N-1}\sum_{k=1}^{N}((\boldsymbol{x}^k)_i - \bar{x_i})((\boldsymbol{x}^k)_j - \bar{x_j}), \end{aligned}$$

where $(\boldsymbol{x}^k)_i$, $(\boldsymbol{x}^k)_j$ are the i^{th} and j^{th} attributes of the k^{th} object, respectively. This use of superscript and subscript is never ideal but is quite common in multivariable analysis.

N.B. $covariance(x_i, x_i) = variance(x_i)$. Thus, the covariance matrix has the variances of the attributes along the diagonal.

Example 2.8. *Suppose we have a small data set* $\{\boldsymbol{x}^1, \boldsymbol{x}^2\}$ *where* $\boldsymbol{x}^1 = (4, 2)$ *and* $\boldsymbol{x}^2 = (6, 3)$. *This will give us a* 2×2 *covariance matrix, where*

$$S = \begin{pmatrix} var(\boldsymbol{x}^1) & cov(\boldsymbol{x^1}, \boldsymbol{x^2}) \\ cov(\boldsymbol{x^2}, \boldsymbol{x^2}) & var(\boldsymbol{x^2}) \end{pmatrix}.$$

First, we need to compute the means:

- $\bar{x_1} = \frac{4+6}{2} = 5$;
- $\bar{x_2} = \frac{2+3}{2} = \frac{5}{2}$.

Next, we need to compute the variance of each:

- $var(\boldsymbol{x}^1) = \frac{1}{1}((4-5)^2 + (6-5)^2) = 2$;
- $var(\boldsymbol{x}^2) = \frac{1}{1}((2-\frac{5}{2})^2 + (3-\frac{5}{2})^2) = \frac{1}{2}$.

Finally, we compute the off-diagonal elements of S*:*

- $cov(\boldsymbol{x}^1, \boldsymbol{x}^2) = \frac{1}{1}[(4-5)(2-\frac{5}{2}) + (6-5)(3-\frac{5}{2})] = 1$;
- $cov(\boldsymbol{x}^2, \boldsymbol{x}^1) = \frac{1}{1}[(2-\frac{5}{2})(4-5) + (3-\frac{5}{2})(6-5)] = 1$.

Putting this all together, the covariance matrix is given by,

$$S = \begin{pmatrix} 2 & 1 \\ 1 & \frac{1}{2} \end{pmatrix}.$$

Exercise: Consider the data set $\{\boldsymbol{x}^1 = \begin{pmatrix} 92 & 80 \end{pmatrix}^T, \boldsymbol{x}^2 = \begin{pmatrix} 60 & 30 \end{pmatrix}^T, \boldsymbol{x}^3 = \begin{pmatrix} 100 & 70 \end{pmatrix}^T\}$. Show that the covariance matrix for this data set is

$$S = \begin{pmatrix} 448 & 520 \\ 520 & 700 \end{pmatrix}.$$

The covariance of two attributes is a measure of the degree to which two attributes vary together and depends on the magnitudes of the variables. A value near 0 indicates that two attributes do not have a (linear) relationship, but it is not possible to judge the degree of relationship between two variables by looking only at the value of the covariance. Because the correlation of two attributes immediately gives an indication of how strongly two attributes are (linearly) related, correlation is preferred to covariance for data exploration. We define the *correlation matrix* R as follows: the ij^{th} entry of R is the correlation between the i^{th} and j^{th} attributes of the data. If x_i, x_j are the i^{th} and j^{th} attributes, respectively, then

$$r_{ij} = correlation(x_i, x_j) = \frac{cov(x_i, x_j)}{\sigma_i \sigma_j},$$

where σ_i, σ_j are the standard deviations of x_i, x_j, respectively. The diagonal entries of R are $correlation(x_i, x_i) = 1$, while the other entries are between -1 and 1.

2.2.3 Graphic Displays

Graphic displays of basis statistical descriptions include *quantile plots, quantile-quantile plots, histograms* and *scatter plots.* Such graphs allow for visual inspection of the data and is useful for data preprocessing. Note that the first three of these show univariate distributions (i.e. data for one attribute), while scatter plots show bivariate distributions (i.e. data involving two attributes). In the interest of keeping things succinct, we will just briefly discuss quantile plots and scatter plots.

- Quantile plots

A quantile plot is a simple and effective way to have a first look at a univariate data distribution. It displays all of the data for the given attribute which allows the user to assess both the overall behaviours and unusual occurrences, and also plots quantile information. Let $\{x_1, \ldots, x_N\}$ be the data sorted in increasing order (i.e. $x_1 = min$, $x_N = max$). Each observation of some ordinal numeric attribute X is paired with a percentage f_i which indicates that approximately $f_i \times 100\%$ of the data are below the value x_i. Note that we say 'approximately' because there may not be a value with exactly a fraction f_i of the data below x_i. Note that the 0.25 percentile corresponds to the quartile Q_1, the 0.50 percentile is the median and the 0.75 percentile is Q_3.

We let $f_i = \frac{i-0.5}{N}$. These numbers increase in equal steps of $\frac{1}{N}$, ranging from $\frac{1}{2N}$ (which is slightly above 0) to $1 - \frac{1}{2N}$ (which is slightly below 1). On a quantile plot, x_i is graphed against f_i. This allows us to compare different distributions based on their quantiles. For example, given the quantile plots of sales data for two different time periods, we can compare their Q_1, median, Q_3 and other f_i values at a glance.

Example 2.9. *Consider the following unit price data for items sold at an electronics store.*

A set of unit price data	
Unit price ($)	*Count of items sold*
40	*275*
43	*300*
47	*250*
-	-
74	*460*
75	*515*
78	*540*
-	-
115	*320*
117	*270*
120	*350*

The above can be represented by a quantile plot, as shown in Figure 2.4.

- Scatter plots

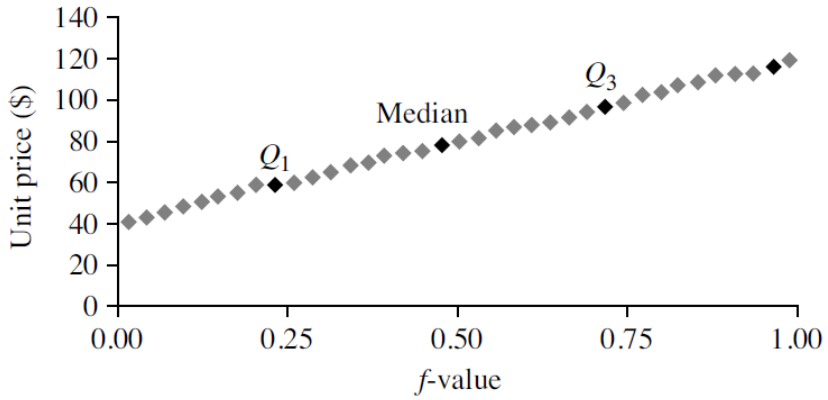


Figure 2.4: A quantile plot for unit price data in Example 2.9. [6]

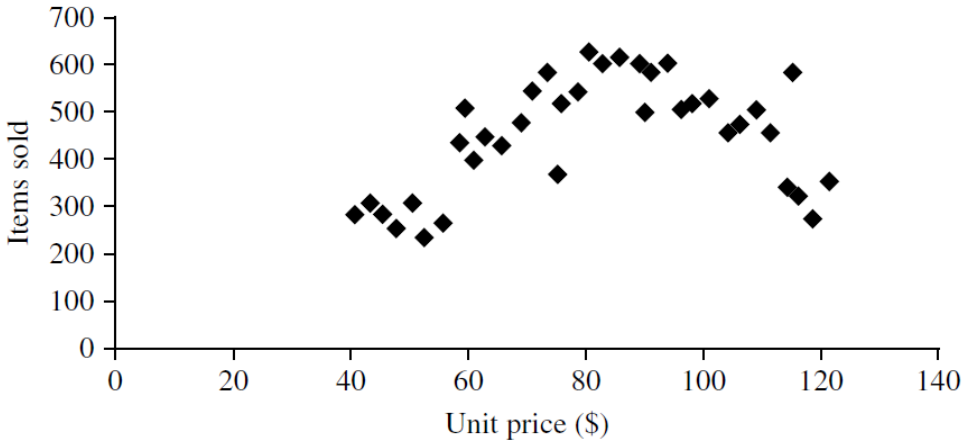


Figure 2.5: A scatter plot for the data in Example 2.9. [6]

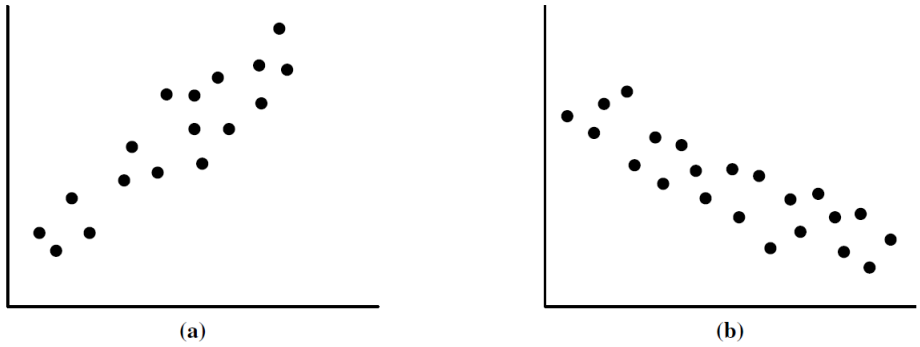


Figure 2.6: Scatter plots can be used to find (a) positive, or (b) negative correlations between attributes. [6]

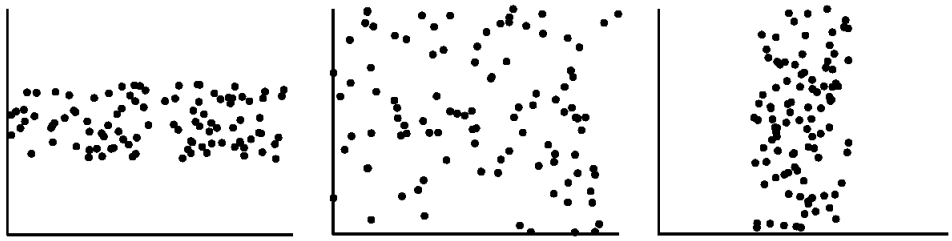


Figure 2.7: Three cases where there is no observed correlation between the two plotted attributes in each of the data sets. [6]

These are incredibly effective at finding relationships/patterns/trends between two numeric attributes. To construct a scatter plot, each pair of values is treated as a pair of coordinates and plotted as plots in the plane. For example, in Figure 2.5, we see a scatter plot for the set of data given in Example 2.9.

When we first get bivariate data, we generally want to have a rough idea of any potential clustering or outliers, as well as any possible correlation relationships. For this, scatter plots are useful in that they are simple and easy to construct. Note that two attributes X, Y are correlated if one attribute implies the other. We can have positive, negative or null (uncorrelated) correlations. In Figures 2.6 and 2.7 we see some examples of these. If the plotted points pattern slopes from bottom left to top right, this means that the values of X increase as the values of Y increase, suggesting a *positive correlation*. If the pattern slopes from top left to bottom right, the values of X increase as the values of Y decrease, suggesting a *negative correlation*. A line of best fit can be drawn to study the correlation between these variables.

2.3 Data Quality

Recall we discussed how data mining algorithms are often applied to data that was collected for another purpose, or for future, unspecified, applications. As such, preventing data quality problems is not typically an option. We cannot always go back to the source, and so instead our focus becomes one of detection and correction, and/or of using algorithms that tolerate poor data quality. The former is often called *data cleaning*.

Errors and other issues with data arise due to a multitude of factors, such as: human error; limitations of measuring devices; flaws in the data collection process. These can result in values, or even entire data objects, to be missing, inaccurate, or even duplicated. A typical example of this might be two different records for the same person who has recently lived at two different addresses. That same person may also have inconsistencies in their health records, where their height is recorded as 2m, but weight only 2kg.

To combat this, we will briefly discuss aspects of data quality related to data measurement (noise, artefacts, bias, precision, accuracy etc.) and collection (outliers, missing and inconsistent values, duplicate data etc.). In a rough sense, data have quality if they satisfy the requirements of their intended use. To say more, we first need to introduce some definitions.

Definition 2.10. ***Measurement error*** *refers to any problem resulting from the measurement process.*

A common problem is that the value recorded differs from the true value. The extent of this difference determines how crucial the measurement error is.

Definition 2.11. *For continuous attributes, the* ***error*** *is the numerical difference of the measured and true value.*

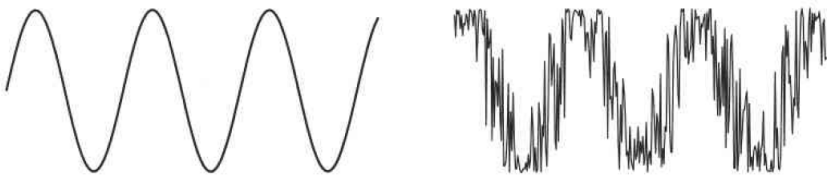
The term ***data collection error*** *refers to errors arising specifically from the data collection process.*

e.g. Omitting data objects/attribute values, and inappropriately including a data object are both examples of data collection errors. These could arise when studying animals. If there is a similar species in the data collection region, then members of the target species could be confused with the similar species leading to a data object not being collected, or the similar species could be confused with the target species leading to an inaccurate recording.

Of course, when working in a given field, certain errors are expected (e.g. musical notation errors when entering data manually) and so subject specific solutions exist. We will not focus on these and will instead look at general types of errors.

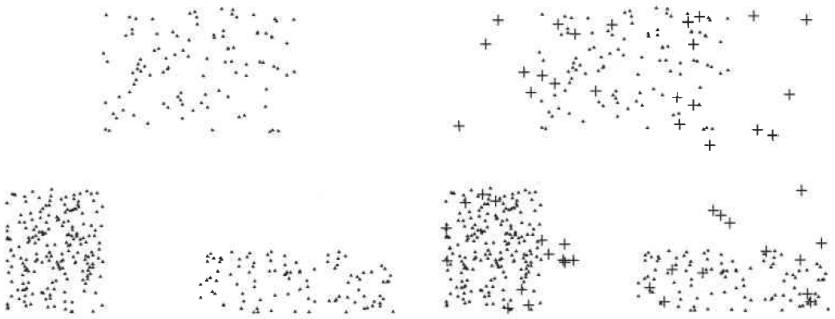
2.3.1 Data Measurement

The first we discuss is *noise*. Noise is a random error or variance in a measured variable and typically involves the distortion of a value or the addition of spurious objects. Noise has two main sources: implicit errors introduced by measurement tools (e.g. different types of sensors); and random errors introduced by batch processes or experts when the data are gathered (e.g. in a document digitalization process). In Figure 2.8, we see a time series



(a) Time series. (b) Time series with noise.

Figure 2.8: Noise in a time series context. [2]



(a) Three groups of points. (b) With noise points (+) added.

Figure 2.9: Noise in a spatial context [2]

before and after it has been disrupted by random noise. In Figure 2.9, a set of data points has some noise added (indicated by the '+'s). Notice that some of the noise points are intermixed with the non-noise points. With enough additions, we could entirely lose the groupings. This can significantly affect classification strategies.

Given the importance of classification, and the difficulty presented by noise, there are lots of resources specifically aimed at overcoming this. One such resource is [10]. You may also want to look at the following: `https://sci2s.ugr.es/noisydata`. In this, the type of noise is discussed, as well as the use of noise filters.

Some strategies to 'smooth' out data (i.e. remove noise) is as follows:

- Binning

  These methods smooth a sorted data value by consulting its 'neighbourhood', i.e. the values around it. The sorted values are distributed into a number of 'bins'. As these methods consult neighbourhoods, they perform what is called *local* smoothing. Consider the following example. We have the data for a price attribute which has been sorted:

  4, 8, 15, 21, 21, 24, 25.

Next, we partition into *equal-frequency* bins of size 3:

$Bin\ 1$	4, 8, 15
$Bin\ 2$	21, 21, 24
$Bin\ 3$	25, 28, 34.

In a *smoothing by bin means* method, each value in a bin is replaced by the mean value of the bin. So, we end up with the following:

$Bin\ 1$	9, 9, 9
$Bin\ 2$	22, 22, 22
$Bin\ 3$	29, 29, 39.

One can likewise use a *smoothing by bin medians* method or *smoothing by bin boundaries*. In this case, each bin value is replaced by the closest boundary value (with one of max/min chosen beforehand for cases where values are equidistant). In general, the larger the width, the greater the effect of the smoothing. Binning is also used as a discretisation technique (see later).

- Regression

  Data smoothing can be done by regression, a technique that conforms data values to a function. Linear regression involves finding the 'best' line to fit two attributes (or variables) so that one attribute can be used to predict the other. Multiple linear regression is an extension of linear regression, where more than two attributes are involved and the data are fit to a multidimensional surface. See Appendix A for information on this.

- Outlier analysis

  Outliers may be detected by clustering, for example, where similar values are organised into groups (or 'clusters). Intuitively, values that fall outside of the set of clusters may be considered outliers.

On the opposite end of the randomness spectrum is an *artefact*. This can be something like a streak in the same place on a set of photographs. These are deterministic distortions of the data.

In statistics and experimental science, the quality of the measurement process and the resulting data are measured by precision and bias.

Definition 2.12. ***Precision*** *is the closeness of repeated measurements (of the same quantity) to one another.*

Bias *refers to the systematic variation of measurements from the quantity being measured.*

Accuracy *is the closeness of measurements to the true value of the quantity being measured.*

A typical way to measure precision is using the standard deviation of a set of values, whereas bias is measured by taking the difference between the mean of the set of values and the known value of the quantity being measured. Given this, bias can only be determined for objects whose measured quantity is known by means external to the current situation. Accuracy depends on precision and bias, but there is no specific formula for accuracy in terms of these quantities. Instead, it is more of a general term that captures both ideas.

Example 2.13. *Suppose we have a standard laboratory weight with a mass of 1g and we want to assess the precision and bias of a new laboratory scale. We weigh the mass five times and obtain the following five values:*

$$\{1.015, 0.990, 1.013, 1.001, 0.986\}.$$

Keeping the notation from above, we have the following:

$$\begin{aligned} N &= 5 \\ \bar{x} &= 1.001 \\ \sigma^2 &= \tfrac{1}{4}[(1.015-1.001)^2 + (0.990-1.001)^2 + (1.013-1.001)^2 + \\ &\qquad +(1.001-1.001)^2 + (0.986-1.001)^2] \\ &= 0.0001715 \\ \sigma &= 0.013 \end{aligned}$$

So, the bias is $1.001 - 1.000 = 0.001$ *and the precision is* 0.013.

An important aspect of accuracy is the use of *significant digits (s.d.)* The goal is to use only as many digits to represent the result of a measurement/calculation as are justified by the precision of the data. Note that we can often alter the s.d. depending on our need mid-computation. For example, above we used 7 s.d. after the decimal point for σ^2 since, for 3 s.d. we would have $\sigma^2 = 0.000$ which would then lead to an error with σ. In general, you do not want to start rounding until absolutely necessary. What 'absolutely necessary' means is very situation dependent. It could relate to memory constraints, for example, or that we are ready to present our output.

2.3.2 Data Collection

- Outliers or anomalous objects

Roughly, these take one of two forms:

1. data objects that, in some sense, have characteristics which are different from most of the other data objects in the data set; or

2. values of an attribute that are unusual with respect to the typical values for that attribute.

As the above suggests, there is considerable leeway in the definition of an outlier, and the situation is as big a player as the values themselves. It is important to distinguish between the notions of noise and outliers. Unlike noise, outliers can be legitimate data objects, or values that we are interested in detecting. A typical example is that of fraud and network intrusion detection. In these cases, the goal is to find unusual objects or events from among a large number of normal ones.

- Missing values

It is not unusual for an object to be missing one or more attribute values. This could be due to the information not being collected (e.g. a person declining to give their age or weight), an attribute not being applicable to all objects (e.g. a form which has conditional parts that are filled out only if a previous question is answered in a specific way), or data being lost (e.g. when transferring data from one database to another). There are several strategies (and variations on these strategies) for dealing with missing data, each of which is appropriate in certain circumstances. In practice, we can utilise several strategies simultaneously. It should also be noted that which strategy we use largely depends on the data and the knowledge we have about it. Ideally, data sets are accompanied by documentation that describes different aspects of the data, e.g. the documentation may identify several attributes that are important and/or strongly related. If the former, then below we may not want to eliminate such attributes. If the latter, then perhaps we do not need all of the attributes since one is known to imply the other. A typical example of this is sales tax and purchase price. Of course, the documentation may be poor (e.g. it may fail to tell us that missing values are indicated with a -9999). In which case, our analysis may be faulty.

Some of these strategies are as follows:

1. Eliminate data objects or attributes

   This is a simple approach and often effective. We simply eliminate those objects which have missing values, or those which have the majority of missing values. Care must be taken here since even a partially specified data object contains some information. Depending on the number of objects with missing values, we may end up removing too much of our dataset.

   A related strategy is to eliminate attributes that have missing values, or those attributes which have the majority of missing values. Again, this should be done with caution since eliminated attributes may be the ones that are critical to the analysis.

2. Estimate missing values

   Sometimes missing data can be reliably estimated. For example, if we have a time series that changes in a reasonably smooth fashion, but has a few widely scattered missing values, then we can replace these missing values with values estimated from nearby points (interpolation). As another example, consider a data set that has many similar data points. In this situation, the attribute values of the points closest to the point with the missing value are often used to estimate the missing values. If the attribute is continuous, then the average attribute value of the nearest neighbours is used; if the attribute is categorical, then the most commonly occurring attribute value can be

used. For example, consider precipitation measurements that are recorded by ground stations. For areas not containing a ground station, the precipitation can be estimated using values observed at nearby ground stations. As ever, care must be taken. For instance, with the last example we need to use more than just distance to generate replacement values. If, for example, the nearest station (in terms of metres) is at a much higher/lower altitude, then this would not necessarily translate to a similar level of precipitation.

3. Ignore the missing value during analysis

   Many data mining approaches can be modified to ignore missing values. For example, suppose that objects are being clustered and the similarity between pairs of objects needs to be calculated. If one or both objects of a pair have missing values for some attributes, then the similarity can be calculated using only the attributes that do not have missing values. How accurate this approximation is will largely depend on how many attribute values are missing, and what the attributes are. For example, return to the earlier example of a form that has questions which only need answers if earlier questions are answered in a specific way. In this case, if we are expecting the missing values, then we need not worry. Similarly, if we have 1000 data objects relating to patients, and only 12 have not input their weight, this will not have a great effect on the analysis.

- Inconsistent values

Data can often contain inconsistent values. For example, when entering your address, it is not uncommon to enter errors. This could be due to a postcode that technically does not belong to a given city, but locally is thought to (this could be due to boundary changes, for example), or due to a typo being entered (O instead of 0, for example). Such errors could be due to an individual or if the information is being scanned from a handwritten form. Regardless of the cause, such inconsistent values are important to detect and, if possible, correct. Depending on the inconsistency, the difference may be easy to correct. For example, if a postcode/city pair are often entered even if they are not technically correct, then this would be an easy fix to implement. Likewise if a height is entered as a negative, this is clearly an inconsistency. However, if the error is a typo (1 instead of 2, say) then this may be more difficult to detect and/or correct.

Example 2.14. *This example illustrates an inconsistency in actual time series data that measures the* ***sea surface temperature (SST)*** *at various points on the ocean. SST data was originally collected using ocean-based measurements from ships or buoys, but more recently using satellites. For example, see NOAA SST Data.*

To create a long-term data set, both sources of data must be used. However, as the data comes from different sources, the two parts of the data are subtly different. This discrepancy is visually displayed in Figure 2.10 which shows the correlation of SST values between pairs of years. If a pair of years has a positive correlation, then the location corresponding to the pairs of years is coloured white; otherwise, it is coloured black. Note that seasonal variations were removed from the data since otherwise all years would be highly correlated. We note that there is a distinct change in behaviour where the data has been put together in 1983. Years

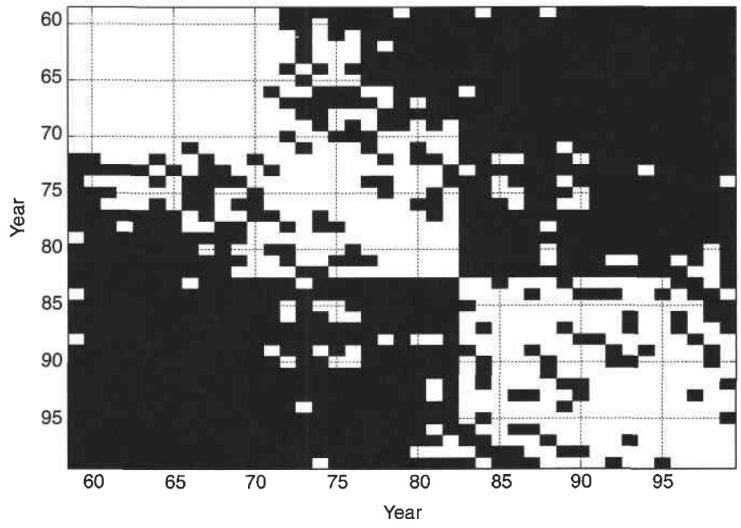


Figure 2.10: Correlation of SST data between pairs of years. White areas indicate positive correlation. Black areas indicate negative correlation. [2]

within each of the two groups (1958-1982, 1983-1999) tend to have a positive correlation with one another, but a negative correlation with years in the other group. This does not mean that this data should not be used, only that the analyst should consider the potential impact of such discrepancies on the data mining analysis.

- Duplicate data

Finally, a data set can include data objects that are duplicates (or near duplicates) of another one. A typical example of this might be a person receiving duplicate mail/notifications due to appearing on a database multiple times under slightly different names (e.g. one with full name, one with middle name(s)). To detect and eliminate such duplicates, two main issues must be addressed. First, if there are two objects that actually represent a single object, then one or more values of corresponding attributes are usually different, and these inconsistent values must be resolved. Second, care needs to be taken to avoid accidentally combining data objects that are similar, but not duplicates, such as two distinct people with identical names[4]. The term *deduplication* is often used to refer to the process of dealing with these issues.

In some cases, two or more objects are identical with respect to the attributes measured by the database, but they still represent different objects. Here, the duplicates are legitimate, but can still cause problems for some algorithms if the possibility of identical objects is not specifically accounted for in their design.

As a final point on the above, note that we also need to be aware of the relevance and timeliness of our data and subsequent data analysis. For the former, we need the available data to contain the necessary information for the application. If we want to build a model

[4]I feel this particular pain more than most.

that predicts accident rates for drivers, but the data does not contain attributes such as age or sex, then our model will naturally be flawed. For the latter, we need to know when our data analysis is required, and when the data was acquired. If we are looking at spending patterns which are temporary, then any analysis delivered after the spending phenomenon has passed is unlikely to be of much use (unless it is expected to be repeated, such as shopping around Christmas time). This time factor has a significant impact on how accurate our data is and what we can do to our data as the whole data cleaning and data preprocessing stages are often time intensive.

2.4 Data Preprocessing

There are several preprocessing techniques and we will only consider five here:

- Aggregation;
- Sampling;
- Dimensionality reduction;
- Discretization;
- Normalisation.

The reader is encouraged to read elsewhere for more detail (e.g. [2] or [6]). Roughly speaking, there are two categories: selecting data objects and attributes for the analysis, or for creating/changing the attributes. In both cases, the goal is to improve the data mining analysis with respect to time, cost and quality.

As a rough overview, *data cleaning* can be applied to remove noise and correct inconsistencies in data. *Data integration* merges data from multiple sources into a coherent data store such as a data warehouse. *Data reduction* can reduce data size by, for instance aggregating, eliminating redundant features, or clustering. Two main strategies include *dimensionality reduction* and *numerosity reduction*. For the former, data encoding schemes are applied so as to obtain a reduced of 'compressed' representation of the original data. A key example of this (for us, at least) will be principal component analysis, but other examples include wavelet transforms, attribute subset selection (e.g. removing irrelevant attributes) and attribute construction (e.g. where a small set of more useful attributes is derived from the original set). In numerosity reduction, the data are replaced by alternative, smaller representations using parametric models (e.g. regression or log-linear models), or nonparametric models (e.g. histograms, clusters, sampling or data aggregation).

Data transformations (e.g. normalisation) may be applied, where data are scaled to fall within a smaller range, such as between 0.0 and 1.0. This can improve the accuracy and efficiency of mining algorithms involving distance measurements (more on this later). For example, consider some customer data that we want to analyse. In this data, we have the attributes *age* and *annual salary*. The annual salary attribute is going to be much larger than values of age. Thus, if the attributes are left unnormalised, the distance measurements taken on annual salary will generally outweigh distance measurements taken on the age attribute.

Discretisation and *concept hierarchy generation* can also be useful, where raw data values for attributes are replaced by ranges or higher conceptual levels. For example, raw values for the age attribute may be replaced by higher-level concepts, such as *youth, adult* or *senior*. These are powerful tools as they enable data mining at multiple abstraction levels.

These techniques are not mutually exclusive, however, and may work together. For instance, data cleaning can involve transformations to correct wrong data, such as by transforming all entries for a data field to a common format.

2.4.1 Aggregation

Aggregation is less well-defined than many other preprocessing tasks. However, it is one of the main techniques used by the database area of Online Analytical Processing (OLAP) and so we include it here. The idea here is to combine two or more objects into a single object. Consider a data set consisting of transactions (data objects) recording the daily sales of products in various store locations (London, Cardiff, Edinburgh, Dublin, and so on) for different days over the course of a year:

Data set containing information about customer purchases					
Transaction ID	Item	Store Location	Date	Price	···
⋮	⋮	⋮	⋮	⋮	
101123	Watch	London	06/09/22	£25.99	···
101123	Battery	London	06/09/22	£5.99	···
101124	Shoes	Cardiff	06/09/22	£75.99	···
⋮	⋮	⋮	⋮	⋮	

One way to aggregate transactions for this data set is to replace all the transactions of a single store with a single storewide transaction. This reduces the hundreds or thousands of transactions that occur daily at a specific store to a single daily transaction, and the number of data objects per day is reduced to the number of stores.

An obvious issue is how an aggregate transaction is actually created. Quantitative attributes such as price are typically aggregated by taking a sum or an average. A qualitative attribute such as an item can be either omitted or summarised in terms of a higher level category (e.g. televisions versus electronics).

Another way to think of the data in the above table is as a multidimensional array, where each attribute is a dimension. From this viewpoint, aggregation is the process of eliminating attributes, such as the type of item, or reducing the number of values for a particular attribute. For example, one can eliminate an attribute such as post-tax income if we already have income and know the tax rate. Another example is if we want to reduce possible values for a date from 365 days. We could choose 12 months or 52 weeks. This type of aggregation is commonly used in OLAP (see [1], [8]).

There are several motivations for aggregation:

- Smaller data sets require less memory and processing time. This enables the use of more expensive data mining algorithms.

- Aggregation can act as a change of scope/scale by providing a high-level view of the data instead of a low-level view. In the above example, aggregating over store locations and months gives us a monthly, per store view of the data instead of a daily, per item view.

- Finally, the behaviour of groups of objects or attributes is often more stable than that of individual objects or attributes. This reflects the statistical fact that aggregate quantities, such as averages or totals, have less variability than the individual values being aggregated. For totals, the actual amount of variation is larger than that of individual objects (on average), but the percentage of the variation is smaller. For means, the actual amount of variation is less than that of individual objects (on average).

All of the above should be weighted against the fact that we may lose interesting details. In the store example, aggregating over months loses information about which day of the week has the highest sales, for example.

2.4.2 Sampling

This is a commonly used approach when selecting a subset of the data objects to be analysed. In statistics, it has long been used for both the preliminary investigation of the data and the final data analysis. It can also be very useful in data mining, however the motivations in data mining often differ from those in statistics. Statisticians use sampling because obtaining the entire set of data of interest is too expensive or time consuming, while data miners usually sample because it is too computationally expensive in terms of the memory or time required to process all of the data. In some cases, using a sampling algorithm can reduce the data size to the point where a better, but more computationally expensive algorithm can be used.

Key Principle: Using a sample will work almost as well as using the entire data set if the sample is representative.

Obvious Question: What makes a sample representative?

This is a difficult question to answer in practice. As a rough starting point, a sample is representative if it has approximately the same property (of interest) as the original set of data. If the mean of the data objects is the property of interest, then a sample is representative if it has a mean that is close to that of the original data. As ever, what we mean by 'close' is context dependent.

Because sampling is a statistical process, the representativeness of any particular sample will vary, and the best that we can do is choose a sampling scheme that guarantees a high probability of getting a representative sample. This involves choosing the appropriate sample size and sampling technique. Focus on the latter first.

- Simple random sampling

  This is the simplest type of sampling and is similar to a lottery. For this to be a truly random sample, every object in the target population must have an equal chance of being chosen.

There are two variations: (1) *sampling without replacement* where, as each object is selected, we remove said object from the target data set; (2) *sampling with replacement* which, as the name suggests, replaces the selected object so that the population does not decrease in size as the sampling proceeds. In general, if the sample is quite small compared to the population, both of these approaches will produce similar samples. As such, it is often easier to use sampling with replacement as the probabilities will remain constant throughout the process.

- Systematic sampling

  This is a variation on simple random sampling and is often used when the target data set is very large. Here, we select names/identifiers systematically from our data set (such as choosing every fourth object in the set). This is sometimes called *quasi-random* sampling because, technically, not every object in the sampling has an equal chance of being chosen. However, usually this is 'random enough' for most samples.

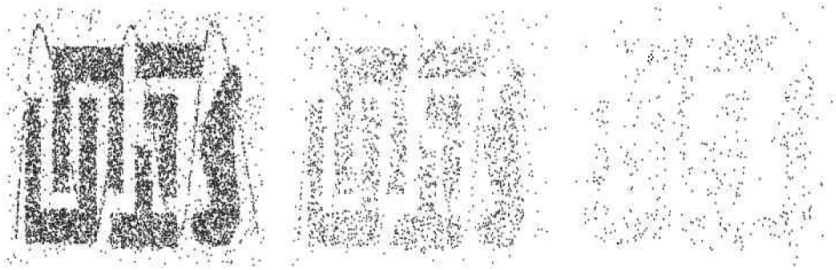
Before moving on, we should discuss some advantages and disadvantages. Obviously, the primary advantage here is that both strategies are quick, easy and time-efficient. This is particularly important when the target population is very large. They are also relatively inexpensive and do not require much information about the data set beyond a given identifier (such as a name or ID number if the data objects are people). Finally, these strategies (assuming our data set hasn't been ordered) will produce random (or near-random) samples which are based on chance.

Of course, there are also some very obvious disadvantages. For one, when the population consists of different types of objects, with widely different numbers of objects, simple random sampling can fail to adequately represent those types of objects that are less frequent. This can cause significant problems. For example, when building classification models for rare classes, it is critical that the rare classes be adequately represented in the sample.

- Stratified sampling

  When we need a sampling scheme that can accommodate differing frequencies for the object types of interest, we can use stratified sampling. This starts by dividing (stratifying) the target population into prespecified groups of objects From here, we have several options. One is called *stratified random sampling* which keeps the idea of sample selection based on chance. The idea is to treat each group as a target for our random sample. From here, we have two options. We can either randomly select from each group, not worrying about their respective sizes. The other way is to randomly select so that the sample of each group is reflective of the overall proportions. For example, a target population of 100 people may include 80 women and 20 men. We take a 10% sample and divide our population into two groups (men and women). As we want 10%, we could take 2 men from the one group, and 8 women from the other group. This would reflect the original proportions. Alternatively, we could randomly select five people from each group (men and women) so that we have our sample of 10.

  Another option is *stratified quota sampling*. This uses the same basic technique as above, but this time we randomly select from the data set and keep track of our



(a) 8000 points (b) 2000 points (c) 500 points

Figure 2.11: Example of the loss of structure with sampling. [2]

choices. It requires knowledge of the proportions in the population. So, for the above example, we would keep selecting at random but once we have two men selected, we will not accept any more men into our sample.

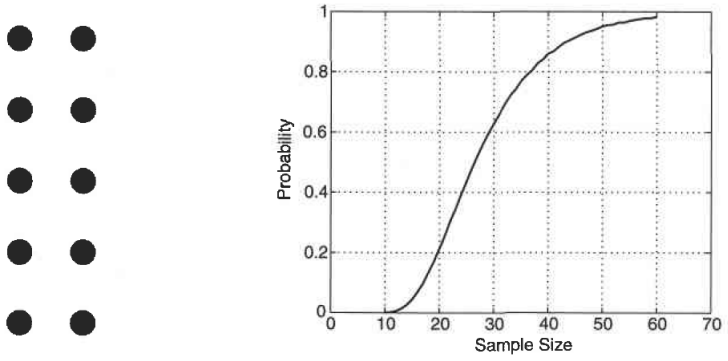
Again, it is worth discussing advantages and disadvantages. The obvious advantage here is that known differences in the target dataset will be accurately reflected in the sample. We can, therefore, be sure our sample will be broadly representative of the target population, at least as far as the situation we are concerned about. Stratified samples can also be relatively small, precisely because it is possible to ensure an accurate reflection of the target population, at least so far as the context in which are interested. In terms of resources, quota samples are usually cheap and quick to construct (this is why they're used by political polling organisations).

One potential drawback is that this technique is not truly random in its sample selection. Not every object in the population has an equal chance of being selected. A more serious problem with stratified samples is that they are critically dependent on the accuracy of information about the data set. If the data set is not accurately modelled (the weighting given to categories, for example, is incorrect) then the sample cannot be representative. Even in situations where accurate information is available, this information may become outdated by the time the data is ready to be analysed.

Example 2.15. *Once a sample technique has been selected, it is still necessary to choose the sample size. Larger sample sizes increase the probability that a sample will be representative, but they also eliminate much of the advantage of sampling. Conversely, with smaller sample sizes, patterns can be missed or erroneous patterns can be detected. See Figure 2.11(a) which shows a data set that contains 8000 two-dimensional points, while Figure 2.11(b)-(c) show samples from this data set of size 2000 and 500, respectively. Although most of the structure of this data set is present in the sample of 2000 points, much of the structure is missing in the sample of 500 points.*

Example 2.16. *Consider the following task:*

Given a set of data consisting of a small number of almost equal sized groups, find at least one representative point for each of the groups. Assume that the objects in each group are highly similar to each other, but not very similar to objects in different groups.



(a) Ten groups of points.

(b) Probability a sample contains points from each of 10 groups.

Figure 2.12: Finding representative points from 10 groups. [2]

This problem can be efficiently solved using sampling. One approach is to take a small sample of data points, compute the pairwise similarities between points, and then form groups that are highly similar. The desired set of points is then obtained by taking one point from each of these groups. To follow this approach, however, we need to determine a sample size that would guarantee, with a high probability, the desired outcome; that is, that at least one point will be obtained from each cluster. See Figure 2.12(a) which shows an idealised set of clusters from which these points might be drawn, and Figure 2.12(b) which shows the probability of getting one object from each of the 10 groups as the sample size runs from 10 to 60. Note that, with a sample size of 20, there is just 20% chance of getting a sample that includes all 10 clusters. With a sample size of 30, this chance increases to 60%.

This last example demonstrates the importance of determining a proper sample size. As can be imagined, determining the proper sample size is difficult. Consequently, adaptive or progressive sampling schemes are sometimes used. These approaches start with a small sample and then increase the sample size until a sample of sufficient size has been obtained. Of course, this requires a way to evaluate the sample to judge if it is large enough. Take, for example, a predictive model. The accuracy of predictive models increase as the sample size increases. At some point, the increase in accuracy will level off and so we want to stop increasing the sample size at this levelling-off point. By keeping track of the change in accuracy of the model as we take larger and larger samples, and by taking other samples close to the size of the current one, we can get an estimate of how close we are to this levelling-off point, and thus stop sampling.

2.4.3 Dimensionality Reduction

Data sets can have a large number of features. Consider a set of documents, where each document is represented by a vector whose components are the frequencies with which each word occurs in the document. In such cases, there are typically thousands or tens of thousands of attributes (components), one for each word in the vocabulary. Another example

is a set of time series consisting of daily closing prices of various stocks over a period of 30 years. In this case, the attributes (which are prices on specific days) again number in the thousands.

In certain cases, dealing with such large dimensions is unnecessary and/or unrealistic. As such, we may want to reduce the number of dimensions so as to make analysis possible, or to utilise other strategies which are only viable for lower dimensions. Indeed, there are many data mining algorithms that work better if the dimensionality is lower. This is partly because dimensionality reduction can eliminate irrelevant features and reduce noise, and partly because of the curse of dimensionality (see below). Another reason we might want to reduce the dimension is that it may make our models more understandable, e.g. by dimension reduction, we may be able to visualise our data more easily. Finally, dimensionality reduction can decrease the amount of time and memory required by the data mining algorithm.

The actual term dimensionality reduction does not refer to a specific method. Instead, it refers to a family of methods, most of which reduce the dimension of a data set by creating new attributes that are a combination of the old attributes[5]. Dimensionality reduction methods include wavelet transforms and principal component analysis, although we will only look at the latter.

- The curse of dimensionality

  This refers to the phenomenon that many types of data analysis becomes significantly harder as the dimensionality of the data increases. In particular, as dimension increases, data becomes increasingly sparse in the space that it occupies. Thus, the data objects we observe are quite possibly not a representative sample of all possible objects. For classification, this can mean that there are not enough data objects to allow the creation of a model that reliably assigns a class to all possible objects. For clustering, the differences in density and in the distances between points (which are critical for clustering) become less meaningful. Consequently, many clustering and classification algorithms have trouble with high-dimensional data, leading to reduced classification accuracy and poor quality clusters.

With the above in mind, we now provide an intuitive introduction to principal component analysis. We will then explore this further in FDS. In a nutshell, PCA uses linear algebra to project the data from a high-dimensional space to a lower-dimensional space. In particular, PCA is used for continuous attributes and finds new attributes (the principal components) that are (1) linear combinations of the original attributes, (2) are orthogonal to each other, and (3) capture the maximum amount of variation in the data.

The basic procedure is as follows:

1. The input data of n dimensional data vectors are normalised. This ensures that attributes with large domains will not dominate attributes with small domains.

2. PCA then computes k orthonormal vectors that provide a basis for the normalised input data. These are unit vectors that each point in a direction orthogonal to the

[5]The reduction of dimensionality by selecting attributes that are a subset of the old is known as feature subset selection or feature selection.

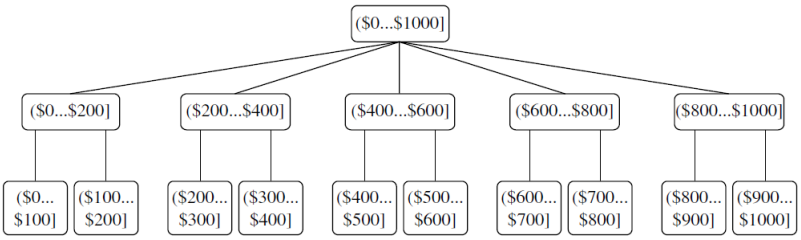


Figure 2.13: A concept hierarchy for the attribute *price*, where an interval $(\$x, \ldots, \$y]$ denotes the range from $\$x$ (exclusive) to $\$y$ (inclusive). [6]

others and are what we call *principal components*. Note that the input data are a linear combination of the principal components.

3. The principal components are sorted in decreasing 'significance' and now serve as new axes for the data. In particular, they provide more information about the variance; that is, the sorted axes are such that the first axis shows the most variance among the data, the second shows the next highest variance, and so on. This information helps identify groups or patterns within the data.

4. Because the components are sorted in decreasing order of 'significance', the data size can be reduced by eliminating the weaker components; that is, those with low variance. Using the strongest principal components, it should be possible to reconstruct a good approximation of the original data.

2.4.4 Discretisation and binarisation

We have already seen two examples of data transformation in the form of aggregation and smoothing (such as binning and regression). Another form of data transformation is discretisation where continuous attributes (such as height or potentially age) are transformed into categorical attributes (such as[6] $\{short, average, tall\}$ or $\{0-10, 11-20, \ldots\}$). We can take this further where continuous and/or discrete attributes may be transformed into one or more binary attributes (binarisation). This is useful for certain classification algorithms, and also for algorithms that find association patterns (which require data that have binary attributes). In general, the labels created in this transformation process can be recursively organised into higher-level concepts, resulting in a *concept hierarchy*. This is more often done for numeric attributes, can be used more generally in certain circumstances. In Figure 2.13, we have a concept hierarchy for the attribute *price*. Of course, more than one concept hierarchy can be defined for the same attribute to accommodate the needs of various users.

We can categorise discretisation techniques based on how the discretisation is performed, such as whether it uses class information or which direction it proceeds (i.e. top-down v bottom-up). If the discretisation process uses class information, then we say it is *supervised discretisation*. Otherwise, it is *unsupervised*. If the process starts by first finding one or a few

[6]The former example consists of what we call *conceptual* labels, while the latter are *interval labels*.

points (called *split points* or *cut points*) to split the entire attribute range, and then repeats this recursively on the resulting intervals, it is called *top-down discretisation* or *splitting*. This contrasts with *bottom-up discretisation* or *merging* which starts by considering all of the continuous values as potential split-points, removes some by merging neighbourhood values to form intervals, and then recursively applies this process to the resulting intervals. As an example, binning is a top-down splitting technique based on a specified number of bins.

Data discretisation and concept hierarchy generation are also forms of data reduction. The raw data are replaced by a smaller number of intervals or concept labels. This simplifies the original data and makes the mining more efficient. The resulting patterns mined are typically easier to understand. Concept hierarchies are also useful for mining at multiple abstraction levels.

We start by looking at a simple technique to binarise a categorical attribute. The process is as follows:

1. Suppose there are m categorical values in a set X.
2. Define an injective map,
$$X \to \{0, 1, 2, \ldots, m-1\} \subset \mathbb{Z}$$
so that each value is assigned a unique integer in the interval $[0, m-1]$. Note, that if the attribute is ordinal, then the order must be maintained by the assignment.
3. Convert each of these m integers to a binary number. Note we need $n = \lceil \log_2(m) \rceil$ binary digits to represent these integers, where we use the ceiling function since not all integers can be represented as a power of 2. We then represent these binary numbers using n binary attributes.

Recall:
$\log_2(N)$ is the exponent value of 2 which gives N, i.e. if $\log_2(N) = k$ then this is the same as saying $2^k = N$. More generally[7], if $\log_a(N) = b$, then $a^b = N$. All the usual log rules apply, regardless of a. So,

- Product: $\log_a(N_1 N_2) = \log_a(N_1) + \log_a(N_2)$.
- Quotient: $\log_a(N_1/N_2) = \log_a(N_1) - \log_a(N_2)$.
- Power: $\log_a(N_1^x) = x \log_a(N_1)$.
- Identity: $\log_a(a) = 1$.

[7]If you are wondering why $\log_e$ is called the natural logarithm, see [4]. In a rough sense, both binary and decimal logs (or any type of log) are of importance only due to our binary and decimal systems, respectively. The former related to computers, while the latter has historical roots and perhaps is linked to the biological fact we have 10 fingers. However, e is a naturally occurring constant and so is more mathematically 'natural'. There are also mathematical reasons, such as solving the differential equation $\frac{dx}{dt} = x$ with initial condition $x = 1$, but we will not discuss this here.

- Log of exponent: $\log_a(a^x) = x$.
- Exponent of log: $a^{\log_a(N)} = N$.

e.g. $\log_2(128) = \log_2(64) + \log_2(2) = \log_2(32) + 2\log_2(2) = \log_2(16) + 3\log_2(2) = 4 + 3 = 7$.

Example 2.17. *Consider a categorical variable with five values:* $\{awful, poor, okay, good, great\}$. *First, define* f *by* $f(awful) = 0$, $f(poor) = 1$, $f(okay) = 2$, $f(good) = 3$, $f(great) = 4$. *Next, since* $\log_2(4) = 2$ *and* $log_2(8) = 3$, *we need three binary variables* x_1, x_2, x_3. *By converting* 0, 1, 2, 3, 4 *to binary, we will get our representations for* x_1, x_2, x_3 *for free. This is shown in the following table:*

Categorical Value	*Integer Value*	x_1	x_2	x_3
awful	*0*	*0*	*0*	*0*
poor	*1*	*0*	*0*	*1*
okay	*2*	*0*	*1*	*0*
good	*3*	*0*	*1*	*1*
great	*4*	*1*	*0*	*0*

Of course, such transformations can cause complications, such as creating unintended relationships among the transformed attributes. For example, in the previous example, we have a correlation between x_2, x_3 arising from the information about the *good* value being encoded using both attributes. Furthermore, association analysis requires asymmetric binary attributes, where only the presence of the attribute (value = 1) is important. For association problems, it is therefore necessary to introduce one asymmetric binary attribute for each categorical value, as shown in the following table:

Categorical Value	Integer Value	x_1	x_2	x_3	x_4	x_5
awful	0	1	0	0	0	0
poor	1	0	1	0	0	0
okay	2	0	0	1	0	0
good	3	0	0	0	1	0
great	4	0	0	0	0	1

If the number of resulting attributes is too large, then other techniques can be used to reduce the number of categorical values before binarisation.

We now move on to discuss discretisation of continuous attributes. This is typically applied to attributes that are used in classification or association analysis. The transformation involves two subtasks: (1) deciding how many categories n to have, and (2) determining how to map the values of the continuous attribute to these categories. In the first step, after the values of the continuous attribute are sorted, they are then divided into n intervals by specifying $n - 1$ split points. In the second, all the values in one interval are mapped to the same categorical value. Thus, the problem of discretisation is one of deciding how many split points to choose and where to place them. The result can be represented either as a set of intervals

$$\{(x_0, x_1], (x_1, x_2], \ldots, (x_{n-1}, x_n)\},$$

where x_0, x_n can be $\pm\infty$, or equivalently, as a series of inequalities

$$x_0 < x \leq x_1, \ldots, x_{n-1} < x < x_n.$$

Consider first the case where class information is not used (i.e. unsupervised discretisation). In this case, relatively simple approaches are common. We consider a few here.

- The *equal width* approach divides the range of the attribute into a user-specified number of intervals, each having the same width. Such an approach can be badly affected by outliers.
- An *equal frequency/equal depth* approach tries to put the same number of objects into each interval. Due to the equal width's weakness with outliers, this method is often preferred.
- A clustering method such as K-means can also be used. We will see this later in the course.
- Visually inspecting the data can sometimes be an effective approach.

Example 2.18. *Consider Figure 2.14(a) which shows data points belonging to four different groups, along with two outliers which are the large dots on each side. The techniques discussed above were applied to discretise the x values of these data points into four categorical values. Note that the point in the data set have a random y component to make it easy to see how many points are in each group. Note also that visual inspection would also work quite well in this case, but is not automatic and can not always be used. As such, we focus on the other three methods.*

First, the split points in each case are represented by dashed lines. Second, if we measure the performance of a discretisation technique by the extent to which different objects that clump together have the same categorical value, then the K-means method performs the best, followed by equal frequency and then equal width. More generally, the best discretisation will depend on the application and often involves domain-specific discretisation. For example, the discretisation of people into low income, middle income and high income is based on economic factors.

Next, we introduce class information (i.e. supervised discretisation) which often produces better classification. This should not be surprising since an interval constructed with no knowledge of class labels often contains a mixture of class labels. A conceptually simple approach is to place the splits in a way that maximises the purity of the intervals, i.e. the extent to which an interval contains a single class label. In practice, however, such an approach requires potentially arbitrary decisions about the purity of an interval and the minimum size of an interval.

To overcome this, we can allow statistics to guide us. Here, we start with each attribute value in a separate interval and create larger intervals by merging adjacent intervals that are similar according to a statistical test (recall, this is called a bottom-up approach). If we wanted a top-down approach then we can start by bisecting the initial values so that the resulting two intervals give minimum entropy. This only needs to consider each value

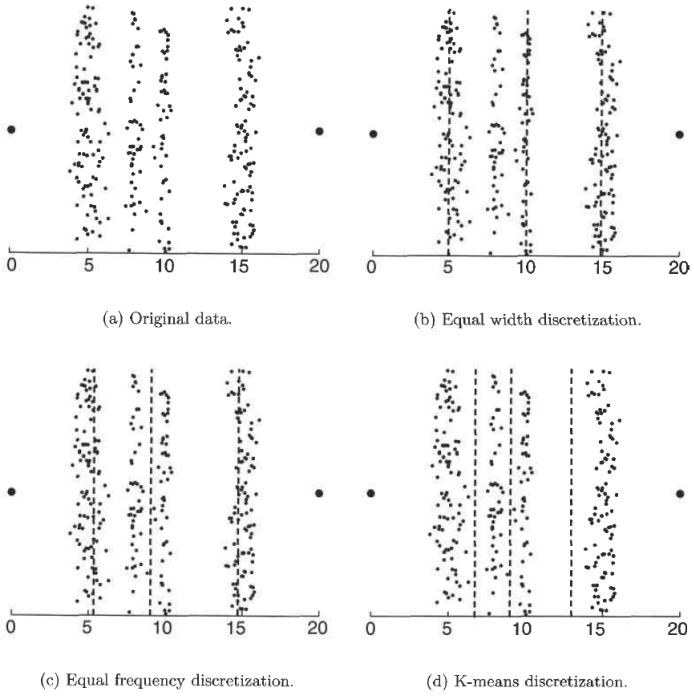


Figure 2.14: Different discretisation techniques. [2]

as a possible split point because it is assumed that intervals contain ordered sets of values. The splitting process is then repeated with another interval, typically choosing the interval with the worst (highest) entropy, until a user-specified number of intervals is reached, or a stopping criterion is satisfied.

Entropy-based approaches are one of the most promising approaches to discretisation, whether bottom-up or top-down, and is used throughout data science. For example, it is used to construct classification trees, forms the basis of *mutual information* which quantifies the relationship between two things, and also shows up in dimension reduction algorithms like t-SNE and UMAP via the Kullback-Leibler Distance. To give a brief explanation of entropy, it is uses to quantify similarities and differences, and does so by quantifying surprise.

The idea is simple enough. Suppose we have a bunch of puppies (say there are 32 of them) and some are black, some are brown. We do not want them to roam all around our house so we gather them up into three separate areas A_1, A_2, A_3. Let's say A_1 has 8 (7 brown, 1 black) puppies, A_2 has 10 (5 black, 5 brown) and A_3 has 14 (4 brown, 10 black). Next, we pick up a a puppy in A_1. The probability of this being a brown puppy is much higher than it being a black puppy because there are 7 brown and just 1 black puppy). Thus, if we picked up a brown puppy, it would not be particularly surprising. Conversely, if we picked up a black puppy, we would be surprised. Likewise, in A_2 (which has an equal number of brown and black puppies), we can pick up a puppy. However, whether it is brown or black we would be equally surprised. Finally, in A_3 we can do the same thing and pick up a puppy. Here there are more black puppies than brown, so the probability of picking a black puppy is higher than picking a brown puppy. As such, if we picked up a black puppy,

we wouldn't be very surprised. In this way, we can think of surprise and probability as being inversely related. Higher the probability, lower the surprise. Lower the probability, higher the surprise.

To actually compute surprise, the obvious thing would be to compute $1/probability$, but this isn't quite right. Go back to A_1 and this time suppose it has no black puppies, i.e. $p(brown) = 1$, $p(black) = 0$. In this case, if we pick up a brown puppy, we should not be surprised at all (i.e. surprise should be 0), but $1/1 = 1 \neq 0$. The answer to this problem is to instead take logarithms (i.e. take $\log(1/prob)$). Now this works for $prob = 1$ since $log(1) = 0$, but there is a problem if $prob = 0$ since $\log(1/0) = \log(1) - \log(0)$ and $\log(0)$ is undefined. Fortunately this isn't too bad since this scenario codes for an event that can never happen. It also means that $\log(n)$ for n very close to 0 is a large number and so $\log(1/prob)$ is very large (and negative!) meaning we are very surprised. Now, one thing to keep in mind is that we use $\log_2$ when we are working with two outcomes (such as heads/tails, or inside/outside an interval). So, the surprise in this case is $\log_2(1/prob)$.

e.g. If we have a coin where $p(H) = 0.9$ and $p(T) = 0.1$, and we flip this coin 3 times, then the surprise of getting a heads, then tails, then heads is

$$surprise = \log_2(\frac{1}{0.9 \times 0.1 \times 0.9}) = \log_2(1) - 2\log_2(0.9) - \log_2(0.1) = 3.62.$$

In particular, note this is just the sum of individual surprises.

We can, of course, extend this. Suppose we now flipped this coin 100 times. Then the surprise of getting a heads in 100 coin flips is just $(0.9 \times 100) \times 0.15 = (prob \times 100) \times surprise$. Likewise, the probability of getting tails is $(0.1 \times 100) \times 3.32$. If we wanted the total surprise, then we just add the outcome of this. In this case, it is 46.7.

This is where (Shannon's) entropy comes in. If we take the above example and divide by 100 (so we get 0.467), we get the average amount of surprise, per coin toss. This is the entropy of the coin, i.e. the expected surprise every time we flip the coin. In reality, we can simply our formula above since we are multiplying and dividing by 100. In reality,

$$E(surprise) = (0.9 \times 0.15) + (0.1 + 3.32) = 0.467.$$

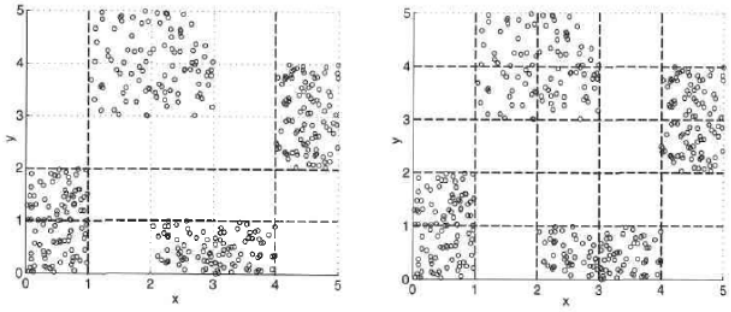
In general,

$$E(surprise) = \sum_x P(x)\log_2(1/P(x)) = -\sum_x P(x)\log_2(P(x)).$$

e.g. Return to the example of the puppies above. The entropy for A_1 is $-(\frac{7/8}{\log 2}(7/8) + \frac{1/8}{\log 2}(1/8)) = 0.54$ Likewise, the entropy for A_2 is 1 and the entropy for A_3 is 0.86.

We can tidy this up and make it more relevant to our purposes. Let k be the number of different class labels, m_i be the number of values in the i^{th} interval of a partition, and m_{ij} be the number of values of j in the interval i. Then the entropy e_i of the i^{th} interval is given by the equation,

$$e_i = \sum_{j=1}^{k} p_{ij}\log_2(p_{ij}),$$



(a) Three intervals (b) Five intervals

Figure 2.15: Discretising x and y attributes for four groups (classes) of points. [2]

where $p_{ij} = \frac{m_{ij}}{m_i}$ is the probability of class j in the i^{th} interval. The total entropy e of the partition is the weighted average of the individual entropies, i.e.

$$e = \sum_{i=1}^{n} w_i e_i,$$

where m is the number of values, $w_i = m_i/m$ is the fraction of values in the i^{th} interval and n is the number of intervals. For us, the intuition of entropy is a measure of the purity of an interval. If an interval contains only values of one class (is perfectly pure) then our expected surprise/entropy is 0. It therefore contributes nothing to the overall entropy. Likewise, if the classes of values in an interval occur equally often (the interval is as impure as possible), then as we saw above, the entropy is at a maximum.

Example 2.19. *The top-down method based on entropy was used to independently discretise both the x and y attributes of the two-dimensional data shown in Figure 2.15. In the first discretisation (showin in Figure 2.15(a)), the x and y attributes were both split into three intervals (the dashed lines indicate split points). In the second discretisation (shown in Figure 2.15(b)), the x and y attributes were both split into five intervals.*

This example illustrates two aspects of discretisation. First, in two dimensions, the classes of points are well-separated, but in one dimension, this is not so. In general, discretising each attribute separately often guarantees suboptimal results. Second, five intervals work better than three, but six intervals do not improve the discretisation much, at least in terms of entropy. Consequently, it is desirable to have a stopping criterion that automatically finds the right number of partitions. For more information on entropy and data mining, see [7].

2.4.5 Normalisation

As discussed earlier, the measurement unit used can affect the data analysis. For example, changing measurement units from meters to inches for *height*, or from kilograms to pounds for *weight*, may lead to very different results[8]. In general, expressing an attribute in smaller units

[8]For an extreme example of this, read about the Mars Climate Orbiter. This is also a lesson for ensuring all measurement data in a data set is written in the same unit.

will lead to a larger range for that attribute, and thus tend to give such an attribute greater effect, or 'weight'. To help avoid dependence on the choice of measurement units, the data should be *normalised* (or *standardised*). This involves transforming the data to fall within a common range (usually [0, 1], or sometimes $[-n, n]$). In this way, normalising attempts to give all attributes equal weight. It is particularly useful for classification algorithms involving neural networks or distance measurements such as nearest-neighbour classification and clustering.

There are many methods for data normalisation. We will briefly discuss three:

- min-max normalisation;
- z-score normalisation;
- normalisation by decimal scaling.

For what follows, let A be a numeric attribute with n observed values $v_1, \ldots, v_n$.

First we look at *min-max normalisation*. This performs a linear transformation on the original data. Suppose that min_A and max_A are the minimum and maximum values of A. Then min-max normalisation maps a value v_i of A to v'_i in the range $[new\_min_A, new\_max_A]$ by computing,

$$v'_i = \frac{v_i - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A.$$

Note that this will preserve the relationships among the original data values and will encounter an 'out-of-bounds' error if a future input case for normalisation falls outside of the original data range for A.

Example 2.20. *Suppose A = income and the values we have are (in order),*

$$£14,000,\ £22,500,\ £33,700,\ £40,900,\ £51,000,\ £124,000.$$

Now suppose we want to map income to the range [0, 1]*. Then, for each v_i, we compute*

$$v'_i = \frac{v_i - 14000}{110000} \times 1 + 0.$$

We summarise the effect of this on the values of A in the following table:

Old Values	*new Values*
14000	*0*
22500	*0.077*
33700	*0.179*
40900	*0.245*
51000	*0.336*
124000	*1*

Next, we look at *z-score normalisation* (also called zero-mean normalisation). Here, the values for A are normalised based on the mean and standard deviation of A. A value v_i is normalised to v_i' by computing

$$v_i' = \frac{v_i - \bar{A}}{\sigma_A},$$

where $\bar{A}$ and σ_A are the mean and standard deviation, respectively. Note that this will not be in the range [0, 1]. If v_i is below the mean, we will get a negative number, and if v_i is above the mean, we get a positive number. The size of these numbers will be determined by the standard deviation. So, if the unnormalised data had a large standard deviation, the normalised values will be closer to 0. This method of normalisation is useful when the actual minimum and maximum of A are unknown, or when there are outliers that dominate the min-max normalisation.

Example 2.21. *Let's use the same values as in the previous example. Then,*

$$\bar{A} = \frac{1}{6}(14000 + 22500 + 33700 + 40900 + 51000 + 124000) = 47683.33$$

and

$$\sigma_A = \sqrt{\frac{1}{5}(7845548333)} = 39611.99.$$

This means that

$$v_i' = \frac{v_i - 47683.33}{39611.99}.$$

This results in the following table:

Old Values	*new Values*
14000	*-0.85*
22500	*-0.64*
33700	*-0.35*
40900	*-0.17*
51000	*0.08*
124000	*1.92*

A variation of this normalisation method replaces the standard deviation by the mean absolute deviation of A. This is defined as,

$$s_A = \frac{1}{n}(|v_1 - \bar{A}| + |v_2 - \bar{A}| + \cdots + |v_n - \bar{A}|).$$

Everything else is the same, i.e.

$$v_i' = \frac{v_i - \bar{A}}{s_A}.$$

The mean absolute deviation, s_A is more robust to outliers than the standard deviation since we are taking absolute values instead of squaring. It is left as an exercise to apply this to the above example.

Finally we look at normalisation by decimal scaling. This normalises by moving the decimal point of values of A. The number of decimal points moved depends on the maximum absolute value of A. A value v_i is normalised to v_i' by computing

$$v_i' = \frac{v_i}{10^j},$$

where j is the smallest integer such that $max(|v_i'|) < 1$.

Example 2.22. *Again, return to the values of the previous two examples. In this case,* $j = 6$ *since that is the smallest* j *such that* $124000/10^j < 1$. *The table will be as follows:*

Old Values	*new Values*
14000	*0.014*
22500	*0.0225*
33700	*0.0337*
40900	*0.0409*
51000	*0.051*
124000	*0.124*

Warning: Normalisation can change the data quite significantly, especially when using z-score normalisation or decimal scaling. It is also necessary to save the normalisation parameters (e.g. the mean and standard deviation if using z-score) so that future data can be normalised in a uniform manner.

2.5 Measures of Similarity and Dissimilarity

The idea of similarity and dissimilarity is a way to formally measure how alike two objects are. There are various ways to do this. For instance, we can use the Jaccard and cosine similarity measures for sparse data such as documents. For non-sparse data such as time series or multi-dimensional points we can use correlation and Euclidean distance. Often, we normalise similarity so that it is measured between 0 and 1 with 0 meaning no similarity and 1 meaning complete similarity, although sometimes the range $[0, \infty)$ is used. In reality, the range can be anything we like (within reason) and different algorithms may require different ranges. As such, we do need to have some understanding of how to transform ranges (see below).

This idea of similarity/dissimilarity is important in a number of data mining techniques, such as clustering, nearest neighbour classification and anomaly detection. Indeed, in many cases, once we have computed the similarities, we do not really need the initial data set any more (see *kernel methods* for more on this). A typical example of how we can employ similarity is in a store where we may want to search for clusters of *customer* objects. This results in groups of customers with similar characteristics, e.g. similar income, area of residence, age etc. Such information can then be used for marketing. Note that we use the term cluster here to mean a collection of data objects in which the objects within a given cluster are similar to other objects in the cluster, and dissimilar to objects in other clusters. Of course, such strategies can also be applied to a range of other problems, such as alerts

for unusual spending patterns or even in diagnosing a patient. In this case, the data object is a patient and they are assigned a class label relating to a diagnosis based on its similarity to other objects in the model.

Of course, similarity and dissimilarity are related (one can perform a simple transformation on $[0, 1]$ by $d = 1 - s$ or on $[-1, 1]$ by $d = -s$, say). Sometimes, it is even beneficial/necessary to convert a measure of similarity to one of dissimilarity. As such, we often work with the idea of *proximity*[9].

As a final point, we note that we can represent similarity/dissimilarity by a similarity/dissimilarity matrix (it is usual to work with the latter). Suppose we have m objects and each has n attributes. As we have seen, we can represent each object as a row of a data matrix,

$$\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{pmatrix}$$

We can now measure the dissimilarity (or difference) between objects i and j, denoted $d(i, j)$. In general, we want $d(i, j)$ to be non-negative and 0 when objects i and j are completely similar, and we want $d(i, j) = d(j, i)$. Given this, we end up with a symmetric matrix (i.e. $A^T = A$) written as follows[10]:

$$\begin{pmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & & \ddots & \\ d(m,1) & d(m,2) & \cdots & \cdots & 0 \end{pmatrix}.$$

2.5.1 Transformations

As mentioned above, it is often necessary to transform the range over which we measure similarity/dissimilarity. The simplest type of example is one where we want to transform *finite* sets. For instance, we have a range $[1, 10]$ and want to change this to $[0, 1]$, say. For simplicity, we suppose 1 means not at all similar and 10 means completely similar. In this case, we can define

$$f : [1, 10] \to [0, 1] \text{ by } f(s) = \frac{s-1}{9}.$$

More generally, if we want to go from $[m, M]$ to $[0, 1]$, then we define

$$g : [m, M] \to [0, 1] \text{ by } g(s) = \frac{s-m}{M-m}.$$

Likewise, we can perform transformations on finite sets for dissimilarity.

[9]Frequently, the term *distance* is used, but this can often refer to a specific class of dissimilarities. As such, we will try to avoid it here.

[10]We have ignored entries above the main diagonal for ease.

Key Point: These transformations preserve the relative distances between points. So, if $x_1\, x_2$ are twice as far apart as points x_3, x_4, then $g(x_1)$, $g(x_2)$ are twice as far apart as $g(x_3)$, $g(x_4)$.

While the above works nicely for finite ranges, problems can occur if the initial range is infinite, for example $[0, \infty)$. In this case, there is no maximum (what we labelled M) for us to define the above formula. A possibly not-so-nice consequence[11] of this is that nice relationships will not be preserved under transformation, such as distance. For example, suppose we measure dissimilarity and want to go from $[0, \infty)$ to $[0, 1]$ using the following transformation,

$$h : [0, \infty) \to [0, 1] \ , \ d \mapsto \frac{d}{d+1}.$$

In this case, 0, 0.5, 2, 10, 100 and 1000 are mapped to 0, $\frac{1}{3}$, $\frac{2}{3}$, $\frac{10}{11} \approx 0.9$, $\frac{100}{101} \approx 0.99$ and $\frac{1000}{1001} \approx 0.999$, respectively. Quite clearly, we see that distances are not preserved and in fact, larger values on the original dissimilarity scale are compressed into the range of values near 1.

Another point to keep in mind is that by transforming to $[0, 1]$, we may lose certain meanings that the original range had. For example, if the initial range is $[-1, 1]$, by mapping to $[0, 1]$ (by taking absolute values, say), we may lose information that the sign carries.

2.5.2 Proximity Measures for Nominal Attributes

Recall that a nominal attribute is one which involves equality and inequality (i.e. they only convey information about the distinctness of objects). Examples include postcodes, employee ID, and so forth. First, suppose our objects are described by just the one nominal attribute which can take N different states. In this case, it is typical to define similarity to be 1 if the attribute values match, and 0 otherwise. Likewise, dissimilarity could be matched the opposite way, i.e. 0 if the attribute values match and 1 if they do not.

Next, suppose we have multiple nominal attributes. In this case, we can define dissimilarity between two objects x_i, x_j based on the ratio of mismatches,

$$d(i, j) = \frac{p - m}{p},$$

where m is the number of matches (i.e. the number of attributes for which x_i, x_j are in the same state), and p is the total number of attributes describing the objects. Weights can also be assigned to increase the effect of m or to assign greater weight to the matches in attributes having a larger number of states. Of course, we can instead work with similarity using the formula,

$$sim(i, j) = 1 - d(i, j) = \frac{m}{p}.$$

Example 2.23. *Suppose we have the following data in which only the first two columns are available:*

[11] This may be desirable, depending on the application.

Object Identifier	*Test 1 - Nominal*	*Test 2 - Ordinal*	*Test 3 - Numeric*
1	*Code A*	*Excellent*	*45*
2	*Code B*	*Fair*	*22*
3	*Code C*	*Good*	*64*
4	*Code A*	*Excellent*	*28*

In this case $p = 1$ *since we just have the one nominal attribute (Test 1). So,* $d(i, j) = 0$ *if the attributes match, and 1 if they differ. We can now compute the dissimilarity matrix for this example, as follows:*

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}.$$

From this, we can see that all objects are dissimilar except objects 1 and 4.

2.5.3 Proximity Measures for Binary Attributes

Recall that a binary attribute has only one of two states - 0 and 1. Usually, 0 means that the attribute is absent, and 1 means that it is present. For example, the attribute might be *smoker* which describes whether or not a patient is a smoker. In this case, 1 might indicate that the patient smokes, while 0 indicates that the patient does not.

Warning: Treating binary attributes as if they are numeric can be misleading. Mathematically, we are often better off working with finite fields (specifically, $\mathbb{F}_2$).

As such, there are methods specific to binary data which are necessary for computing dissimilarity. One approach involves computing a dissimilarity matrix from the given binary data. Suppose all binary attributes are thought of as having the same weight. In this case, we have the following 2×2 contingency table, where

- f_{11} = number of attributes that equal 1 for both objects.
- f_{10} = number of attributes that equal 1 for object x_i but 0 for object x_j.
- f_{01} = number of attributes that equal 0 for object x_i but 1 for object x_j.
- f_{00} = number of attributes that equal 0 for both objects.

		Object x_j		
		1	0	Sum
Object x_i	1	f_{11}	f_{10}	$f_{11} + f_{10}$
	0	f_{01}	f_{00}	$f_{01} + f_{00}$
	Sum	$f_{11} + f_{01}$	$f_{10} + f_{00}$	p

- Simple Matching Coefficient

A commonly used similarity coefficient is the *simple matching coefficient* (SMC), which is defined as follows:

$$SMC = \frac{Number\ of\ matching\ attribute\ values}{Number\ of\ attributes} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}.$$

This measure counts both presences and absences equally. Consequently, the SMC could be used to find students who had answered questions similarly on a test that consisted only if true/false questions.

- Symmetric Binary Dissimilarity

  Symmetric binary attributes are those in which each state is equally valuable. Dissimilarity that is based on symmetric binary attributes is called *symmetric binary dissimilarity*. If objects x_i, x_j are described by symmetric binary attributes, then the dissimilarity between x_i, x_j is given by,

$$d(i,\, j) = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}}.$$

- Asymmetric Binary Dissimilarity

  For asymmetric binary attributes, the two states are *not* equally important (for instance, in the case where positive (1) and negative (0) are outcomes of a disease test). Given two asymmetric binary attributes, the agreement of two 1s (a positive match) is then considered more significant than that of two 0s (a negative match). Therefore, such binary attributes are often considered 'monary' (having one state). The dissimilarity based on these attributes is called *asymmetric binary dissimilarity*, where the number of negative matches f_{00} is considered unimportant and is thus ignored:

$$d(i,\, j) = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01}}.$$

- Jaccard Coefficient

  Complementing this, we can measure the difference between two binary attributes based on the notion of similarity instead of dissimilarity. This can be computed as,

$$sim(i,\, j) = \frac{f_{11}}{f_{11} + f_{10} + f_{01}} = 1 - d(i,\, j).$$

  This coefficient is called the *Jaccard coefficient* and is popular in the literature.

Example 2.24. *Suppose that we have the following patient record table containing the attributes name, sex, fever, cough, Test 1, Test 2, Test 3, Test 4:*

name	*sex*	*fever*	*cough*	*Test 1*	*Test 2*	*Test 3*	*Test 4*
Tom	*M*	*Y*	*N*	*P*	*N*	*N*	*N*
Dick	*M*	*Y*	*Y*	*N*	*N*	*N*	*N*
Mary	*F*	*Y*	*N*	*P*	*N*	*P*	*N*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Here, name is an object identifier, sex is a symmetric attribute and the remaining attributes are asymmetric binary. For the asymmetric values we let Y (yes) and P (positive) be set to 1, and the value N (no or negative) be set to 0. Suppose that the distance between objects (patients) is computed based only on the asymmetric attributes. Then, the distance between each pair of the three patients (Tom, Dick and Mary) is as follows:

$$\begin{aligned} d(Tom,\, Dick) &= \tfrac{1+1}{1+1+1} = \tfrac{2}{3} \\ d(Tom,\, Mary) &= \tfrac{0+1}{2+0+1} = \tfrac{1}{3} \\ d(Dick,\, Mary) &= \tfrac{1+2}{1+1+2} = \tfrac{3}{4}. \end{aligned}$$

Example 2.25. *Suppose we want to compute the Jaccard and SMC similarity measures for the following two binary vectors,*

$$\boldsymbol{x} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{y} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

So, $f_{11} = 0$, $f_{01} = 2$, $f_{10} = 1$, $f_{00} = 7$. This means,

$$\begin{aligned} SMC &= \tfrac{0+7}{2+1+0+7} = 0.7 \\ J &= \tfrac{0}{2+1+0} = 0. \end{aligned}$$

2.5.4 Dissimilarity of Numeric Data

The most popular distance measure is the *Euclidean distance* measure, which is a simple generalisation of the well-known Pythagorean Theorem. It can also be thought of as 'straight line' distance or 'as the crow flies'. Suppose we are in $\mathbb{R}^n$ (that is, we are representing elements as n-dimensional vectors. Then we can write,

$$\begin{aligned} x_i &= \begin{pmatrix} x_{i1} & x_{i2} & \cdots & x_{in} \end{pmatrix}^T \\ x_j &= \begin{pmatrix} x_{j1} & x_{j2} & \cdots & x_{jn} \end{pmatrix}^T. \end{aligned}$$

The Euclidean distance between the objects x_i, x_j can then be defined as,

$$d(i,\, j) = \sqrt{\sum_{k=1}^{n} (x_{ik} - x_{jk})^2} = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{in} - x_{jn})^2}.$$

Recall that we can write square roots using indices as $\sqrt{x} = x^{\frac{1}{2}}$. In this sense, we can generalise the Euclidean distance to the *Minkowski* distance measure given as follows:

$$d(i,\, j) = \left(\sum_{k=1}^{n} |x_{ik} - x_{jk}|^r\right)^{\frac{1}{r}},$$

for some parameter r. Of course, when $r = 2$, this is just the usual Euclidean metric.

Important Point: The r parameter should not be confused with the number of dimensions/attributes n.

The three most common examples of Minkowski metrics are as follows:

- $r = 1$, City block/Manhattan/Taxi-cab distance (also called L_1 distance/norm).

  This is so named because it is the distance in blocks between any two points in a city, such as 2 blocks down and 3 blocks over for a total of 5 blocks. It is written,

  $$d(i,\, j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{in} - x_{jn}|.$$

  A common example is the *Hamming distance*, which is the number of bits that is different between two objects that have only binary attributes.

- $r = 2$, Euclidean distance. Again, sometimes called the L_2 distance/norm.

  We can generalise the formula above. If each attribute is assigned a weight according to its perceived importance, the *weighted* Euclidean distance can be computed as

  $$d(i,\, j) = \sqrt{w_1(x_{i1} - x_{j1})^2 + w_2(x_{i2} - x_{j2})^2 + \cdots + w_n(x_{in} - x_{jn})^2}.$$

  Clearly, if each $w_i = 1$, then this is just the same Euclidean distance given above.

- $r \to \infty$, Supremum distance/L_∞ norm.

  This is the maximum distance between any attribute of the objects. We can write the L_∞ norm as,

  $$d(i,\, j) = lim_{r\to\infty} \left(\sum_{k=1}^{n} |x_{ik} - x_{jk}|^r\right)^{\frac{1}{r}}.$$

Example 2.26. *Consider the diagram shown in Figure 2.16, in which we have two objects x_1, x_2. If we want to work out the Euclidean distance, we compute $\sqrt{(3-1)^2 + (5-2)^2} = \sqrt{13}$. By contrast, the Manhattan distance is given by $|3-1| + |5-2| = 2 + 3 = 5$. Finally, the supremum distance is taken by solving,*

$$\lim_{r\to\infty} (2^r + 3^r) = 3.$$

While is is trickier to understand, an intuitive way to think of it is that as r gets large, $2^r + 3^r$ starts to look more and more like 3^r. This leaves us with $(3^r)^{\frac{1}{r}} = 3$. WARNING! This is very hand wavy and you should not think you can always do this.

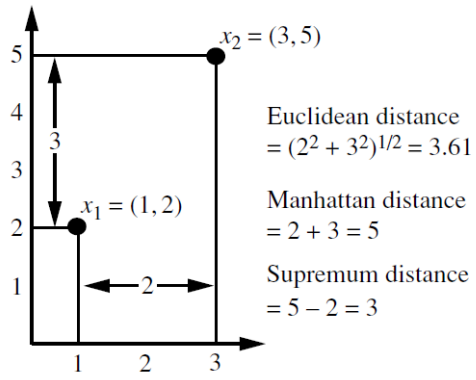


Figure 2.16: Euclidean, Manhattan and supremum distances between two objects. [6]

While you should be very careful when limits are involved, in these cases with finitely many attributes and everything reasonably well behaved, we can make the following statement,

$$\lim_{r\to\infty}\left(\sum_{k=1}^{n}|x_{ik}-x_{jk}|^r\right)^{\frac{1}{r}} = max\{|x_{i1}-x_{j1}|,\cdots,|x_{in}-x_{jn}|\}.$$

Example 2.27. *Consider now four points in* $\mathbb{R}^2$ *represented in the following table:*

Point	*x coordinate*	*y coordinate*
x_1	*0*	*2*
x_2	*2*	*0*
x_3	*3*	*1*
x_4	*5*	*1*

We can now compute the L_1, L_2, L_∞ *norms as follows:*

L_1	x_1	x_2	x_3	x_4
x_1	*0*	*4*	*4*	*6*
x_2	*4*	*0*	*2*	*4*
x_3	*4*	*2*	*0*	*2*
x_4	*6*	*4*	*2*	*0*

L_2	x_1	x_2	x_3	x_4
x_1	*0*	*2.8*	*3.2*	*5.1*
x_2	*2.8*	*0*	*1.4*	*3.2*
x_3	*3.2*	*1.4*	*0*	*2*
x_4	*5.1*	*3.2*	*2*	*0*

L_∞	x_1	x_2	x_3	x_4
x_1	*0*	*2*	*3*	*5*
x_2	*2*	*0*	*1*	*3*
x_3	*3*	*1*	*0*	*2*
x_4	*5*	*3*	*2*	*0*

Now, it may have been noticed that above we used terms such as 'metric' and 'norm' without really defining these concepts. Without delving too much into the mathematics, a metric is a generalisation of Euclidean distance. For any set X, we say that a metric on X is a function $d : X \times X \to X$ such that

- $d(x, y) \geq 0$ for all $x, y \in X$. Moreover, $d(x, y) = 0$ if and only if $x = y$. We call this property *positive definiteness* and it generalises the idea that distance 'should' be non-negative and zero only when the two things we are measuring are at the same spot.

- $d(x, y) = d(y, x)$ for all $x, y \in X$. This property is called *symmetry* and reflects the idea that the distance between x and y 'should' be the same as the distance between y and x.
- $d(x, y) \leq d(x, z) + d(z, y)$ for all $x, y, z \in X$. This is called the *triangle inequality* and the idea is that going from x to y should not be greater than going x to some z, and then z to y.

When we have a set X and a metric d, then we say (X, d) is a metric space. A typical example would be $\mathbb{R}^n$ with the Euclidean metric. Note that in the Euclidean plane (i.e. $\mathbb{R}^2$), the triangle inequality is just saying that the length of one side of a triangle is less than the sum of the lengths of the other two sides.

The idea of a norm is related to the idea of a metric. In a nutshell, if a metric generalises distance, the norm generalises length. So take a vector space, if we have two vectors x, y, then we can use the metric to compute their distance. If we want to compute the length of the vectors, however, then we compute the distance from x to the origin. So, in $\mathbb{R}^n$, the norm is

$$||x|| = d(x, 0) = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}.$$

If we have a norm, we can even go the other way by defining a metric d by $d(x, y) = ||x - y||$.

If we have a vector space with a norm, then we often call this a *normed vector space*. At this point, it may be natural to wonder why we distinguish between a metric space and a normed space, or a metric and a norm. The reason for this perhaps goes too far into the realm of mathematics, but in a general sense, a normed vector space is always a metric space, but a metric space may not be a normed vector space. For instance, the metric space may not have a vector space structure, and so the idea of length in this way may make no sense (there may be no origin).

As a final comment on such distance measures, note that not all measures will be metrics. For example, we can define a distance measure based on the notion of the difference of two sets. Take two sets X, Y. As seen in FDS, $X \backslash Y$ is the set of elements in X that are *not* in Y. So, if $X = \{1, 2, 3\}$ and $Y = \{2, 3\}$, then $X \backslash Y = \{1\}$ and $Y \backslash X = \emptyset$. We can define the distance d between two sets X, Y as,

$$d(X, Y) = |X \backslash Y|,$$

i.e. the number of elements in X that are not in Y. As seen in the simple example above, $d(X, Y) \neq d(Y, X)$, in general, and so does not satisfy the symmetry property in the definition of a metric. It also does not satisfy the triangle inequality or the second part of the positive definite property.

2.5.5 Proximity Measures for Ordinal Attributes

Recall, an ordinal attribute is one in which we also take order into account (compared to nominal). For simplicity, suppose we have just the one attribute. For example, perhaps this attribute measures the quality of a product on the scale {*poor, fair, okay, good, amazing*}. It is reasonable to say that a product P_1 which is rated *amazing* is closer to a product P_2 which is rated *good* than it would be to a product P_3 which is rated *okay*. To make this

quantitative, the values of the ordinal attribute are mapped to successive integers (often starting at 0 or 1), e.g. $\{poor = 0, fair = 1, okay = 2, good = 3, amazing = 4\}$. In this case, $d(P_1, P_2) = 4 - 3$. If we wanted dissimilarity to fall between 0 and 1, then we could set $d(P_1, P_2) = \frac{4-3}{4} = 0.25$. As ever, a similarity attribute can then be defined as $s = 1 - d$.

Of course, this does assume equal intervals between successive values of the attribute, and this need not be the case (if it did, we would be dealing with an interval or ratio attribute). This leads to questions such as: *is the difference between the values fair and good really the same as that between the values okay and amazing?*. The answer is not easy to come by and one would say it is probably not. Nevertheless, in practice our options are limited and so, in the absence of more information, this is the standard approach for defining proximity between ordinal attributes.

More generally, the treatment of ordinal attributes is similar to that of numeric attributes when computing dissimilarity between objects. Suppose that f is an attribute from a set of ordinal attributes describing m objects. The dissimilarity computation with respect to f involves the following steps:

1. The value of f for the i^{th} object is x_{if}, and f has M_f ordered states, representing the ranking $1, \ldots, M_f$. Replace each x_{if} by its corresponding rank $r_{if} \in \{1, \ldots, M_f\}$ (again, can start at 0 if preferred).

2. Since each ordinal attribute can have a different number of states, it is often necessary to map the range of each attribute onto [0, 1] so that each attribute has equal weight. We perform such data normalisation by replacing the rank r_{if} of the i^{th} object in the f^{th} attribute by,
$$z_{if} = \frac{r_{if} - 1}{M_f - 1}.$$

3. Dissimilarity can then be computed using any of the distance measures described above for numeric attributes, using z_{if} to represent the f value for the i^{th} object.

Example 2.28. *Return to the table of data from earlier. This time, we only have the object identifier and the continuous ordinal attribute Test 2:*

Object Identifier	*Test 1 - Nominal*	*Test 2 - Ordinal*	*Test 3 - Numeric*
1	*Code A*	*Excellent*	*45*
2	*Code B*	*Fair*	*22*
3	*Code C*	*Good*	*64*
4	*Code A*	*Excellent*	*28*

There are three states for Test 2: fair, good and excellent. So, using the above notation, $M_f = 3$.

- *Step 1 - replace each value for Test 2 by its rank. The four objects are assigned the ranks* 3, 1, 2, 3, *respectively.*
- *Step 2 - normalise the ranking by mapping rank 1 to 0, rank 2 to 0.5 and rank 3 to 1.*
- *Step 3 - use the Euclidean distance (say) to compute distances. This results in the following dissimilarity matrix:*

$$\begin{pmatrix} 0 & & & \\ 1 & 0 & & \\ 0.5 & 0.5 & 0 & \\ 0 & 1 & 0.5 & 0 \end{pmatrix}.$$

Using this, objects 1 and 2 are the most dissimilar, as are objects 2 and 4 (i.e. $d(2, 1) = d(4, 2) = 1$*).*

2.5.6 Dissimilarity for Attributes of Mixed Types

So far, we have only discussed the dissimilarity between objects described by attributes of the same type (where these types may be either *nominal, symmetric binary, asymmetric binary, numeric* of *ordinal*). However, in many real databases, objects are described by a mixture of attribute types.

One approach to deal with this is to group each type of attribute together and performing separate data mining analysis (e.g. clustering) for each type. This is feasible if these analyses derive compatible results. However, in real applications this is unlikely to be the case. A more preferable approach is to process all attribute types together, performing a single analysis. One such technique combines the different attributes into a single dissimilarity matrix, bringing all of the meaningful attributes onto a common scale of the interval [0, 1].

Suppose that our data set contains n attributes of mixed type. The dissimilarity $d(i, j)$ between objects x_i, x_j is then defined as,

$$d(i, j) = \frac{\sum_{f=1}^{n} \delta_{ij}^{(f)} d_{ij}^{(f)}}{\sum_{f=1}^{n} \delta_{ij}^{(f)}},$$

where

- the indicator $\delta_{ij}^{(f)} = 0$ if either,
  1. x_{if} or x_{jf} is missing (i.e. there is no measurement of attribute f for object x_i or object x_j); or
  2. $x_{if} = x_{jf} = 0$ and attribute f is asymmetric binary.
  3. In any other case, $\delta_{ij}^{(f)} = 1$.
- The contribution of attribute f to the dissimilarity between i and j (i.e. $d_{ij}^{(f)}$) is computed dependent on its type:
  - If f is numeric,
    $$d_{ij}^{(f)} = \frac{|x_{if} - x_{jf}|}{max_r(x_{rf}) - min_r(x_{rf})},$$
    where r runs over all non-missing objects for attribute f.
  - If f is nominal or binary,
    $$d_i^{(f)} = \begin{cases} 0, & \text{if } x_{if} = x_{jf}; \\ 1, & \text{otherwise.} \end{cases}$$

– If f is ordinal, compute the ranks r_{if} and $z_{if} = \frac{r_{if}-1}{M_f-1}$, and treat z_{if} as numeric.

These steps are identical to what has already been seen for each of the individual attribute types. The only difference is for numeric attributes, where we normalise to that the values map to the interval [0, 1]. Thus, the dissimilarity between objects can be computed even when the attributes describing the objects are of different types.

Example 2.29. *Return once again to the following table, this time including all attributes:*

Object Identifier	*Test 1 - Nominal*	*Test 2 - Ordinal*	*Test 3 - Numeric*
1	*Code A*	*Excellent*	*45*
2	*Code B*	*Fair*	*22*
3	*Code C*	*Good*	*64*
4	*Code A*	*Excellent*	*28*

Above, we have already worked out the dissimilarity matrices for each of the individual attributes. The procedures we followed for Test 1 (which is nominal) and Test 2 (which is ordinal) are the same as outlined earlier for processing attributes of mixed types. It therefore remains to compute the dissimilarity matrix for Test 3 (which is numeric) and then utilise this along with our earlier results.

So, we need to compute $d_{ij}^{(3)}$. *Following the case for numeric attributes, we let* $max_r(x_r) = 64$ *and* $min_r(x_r) = 22$. *The difference between the two is used to normalise the values of the dissimilarity matrix. the resulting dissimilarity matrix for Test 3 is,*

$$\begin{pmatrix} 0 & & & \\ 0.55 & 0 & & \\ 0.45 & 1 & 0 & \\ 0.4 & 0.14 & 0.86 & 0 \end{pmatrix}$$

We now use the dissimilarity matrices for each of the three attributes. The indicator $\delta_{ij}^{(f)} = 1$ *for each of the three attributes* f. *We get, for example,*

$$d(3,\, 1) = \frac{1(1) + 1(0.5) + 1(0.45)}{3} = 0.65.$$

The resulting dissimilarity matrix obtained for the data described by the three attributes of mixed type is then,

$$\begin{pmatrix} 0 & & & \\ 0.85 & 0 & & \\ 0.65 & 0.83 & 0 & \\ 0.13 & 0.71 & 0.79 & 0 \end{pmatrix}.$$

From the table, we can intuitively guess that objects 1 and 4 are the most similar, based on their values for Test 1 and Test 2. This is confirmed by the dissimilarity matrix, where $d(4,\, 1)$ *is the lowest value for any pair of different objects. Similarly, the matrix indicates that objects 1 and 2 are least similar.*

2.5.7 Cosine Similarity

Documents are often represented as vectors, where each component (attribute) represents the frequency with which a particular term (word) occurs in the document. Thus, each document is an object represented by what is called a *term-frequency* vector. A simple example is the following table which represents articles on each of the top four teams in the Premier League this season (22/23) after 7 games:

Document	Team	Manager	Yellow Cards	Won	Lost	Drawn	Scored	Conceded	International Break
Article 1	4	7	3	1	4	2	3	2	1
Article 2	2	5	0	2	6	4	2	3	0
Article 3	1	5	0	4	4	1	1	1	2
Article 4	1	8	0	3	3	0	0	0	0

So, in Article 1, there were 7 mentions of the word *manager*, but only 1 mention of *won* and in Articles 2-4 there are no mentions of *Yellow Cards*. Other examples include information retrieval, text document clustering, biological taxonomy, and gene feature mapping. Such data can be highly asymmetric.

Term-frequency vectors are typically very long and *sparse* (i.e. they have many 0 values. As such, much like transaction data, similarity should not depend on the number of shared 0 values because any two documents are likely to 'not contain' many of the same words, and therefore if 0-0 matches are counted, most documents will be highly similar to most other documents. Consequently, a similarity measure for documents needs to ignore 0-0 matches like the Jaccard measure, but also must be able to handle non-binary vectors. Cosine similarity is a common measure and can be used to compare documents or, for example, give a ranking of documents with respect to a given vector of query words. Let $\boldsymbol{x}$, $\boldsymbol{y}$ be two vectors for comparison. Using the cosine measure as a similarity function, we have

$$sim(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}||\,||\boldsymbol{y}||},$$

where

- $||\boldsymbol{x}||$ is the Euclidean norm of $\boldsymbol{x} = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix}^T$ given by $||\boldsymbol{x}|| = \sqrt{x_1^2 + \cdots + x_n^2}$.
- $\boldsymbol{x} \cdot \boldsymbol{y}$ is the scalar product of the two vectors (also called the dot product[12]) and can be written in two ways:

$$\begin{aligned} \boldsymbol{x} \cdot \boldsymbol{y} &= x_1 y_1 + x_2 y_2 + \cdots + x_n y_n \\ &= ||\boldsymbol{x}||\,||\boldsymbol{y}|| \cos(\theta), \end{aligned}$$

  where θ is the angle between $\boldsymbol{x}$ and $\boldsymbol{y}$.

So, the measure computes the cosine of the angle between the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. Recall, a cosine value of 0 means that the two vectors are at 90 degrees to each other (orthogonal)

[12]In the literature, it is also called an inner product. However, as it is not the only inner product on Euclidean space, we will try to ignore that usage.

and have no match. The closer the cosine value is to 1, the smaller the angle and the greater the match between vectors.

N.B. As $\cos(\theta)$ ranges from -1 to 1, we might expect to get negative similarity. This will not happen if we only have non-negative values in our document vectors. However, if we allow negative values in our vectors (dimensionality reduction via PCA might do this, for example), then we can get negative similarity. In this case, the vectors are pointing in opposite directions and we can think of this as 'anti'-similarity.

Note that cosine similarity is not a metric (check this as an exercise), but nevertheless works well for asymmetric attributes since it only depends on components that are both non-zero. In other words, the similarity between the two documents depends only upon the words that appear in both of them.

Example 2.30. *1. Suppose that $\boldsymbol{x}$, $\boldsymbol{y}$ are the first two term-frequency vectors in the above table, i.e.*

$$\begin{aligned} \boldsymbol{x} &= \begin{pmatrix} 4 & 7 & 3 & 1 & 4 & 2 & 3 & 2 & 1 \end{pmatrix}^T \\ \boldsymbol{y} &= \begin{pmatrix} 2 & 5 & 0 & 2 & 6 & 4 & 2 & 3 & 0 \end{pmatrix}^T \end{aligned}$$

Then, $\boldsymbol{x} \cdot \boldsymbol{y} = 89$, $||\boldsymbol{x}|| = \sqrt{109}$ and $||\boldsymbol{y}|| = 7\sqrt{2}$. Putting this all together gives,

$$sim(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}||\,||\boldsymbol{y}||} = \frac{89}{7\sqrt{109}\sqrt{2}} = 0.861\ldots$$

So, if we were using cosine similarity to compare these documents, they would be considered reasonably similar.

2. Suppose that $\boldsymbol{x}$, $\boldsymbol{y}$ are the first two term-frequency vectors in the above table, i.e.

$$\begin{aligned} \boldsymbol{x} &= \begin{pmatrix} 3 & 2 & 0 & 5 & 0 & 0 & 0 & 2 & 0 & 0 \end{pmatrix}^T \\ \boldsymbol{y} &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 \end{pmatrix}^T \end{aligned}$$

Then, $\boldsymbol{x} \cdot \boldsymbol{y} = 5$, $||\boldsymbol{x}|| = 6.48$ and $||\boldsymbol{y}|| = 2.45$. Putting this all together gives,

$$sim(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}||\,||\boldsymbol{y}||} = 0.31$$

So, if we were using cosine similarity to compare these documents, they would be considered reasonably dissimilar.

2.5.8 Correlation

Correlation is frequently used to measure the linear relationship between two sets of values that are observed together. Thus, correlation can measure the relationship between two variables (height and weight), or between two objects (a pair of temperature time series). Correlation is used much more frequently to measure the similarity between attributes since

the values in two data objects can come from different attributes, which can have very different attribute types and scales. There are many types of correlation, and indeed correlation is sometimes used in a general sense to mean the relationship between two sets of values that are observed together. For nominal data, the χ^2 test (chi-square) is popular, while for numeric attributes, Pearson's correlation is popular. For these notes, we will just discuss the latter. Suppose $\boldsymbol{x}$, $\boldsymbol{y}$ are two vectors. Then we define Pearson's correlation as,

$$corr(\boldsymbol{x}, \boldsymbol{y}) = \frac{covariance(\boldsymbol{x}, \boldsymbol{y})}{standard\ deviation(\boldsymbol{x}) \times standard\ deviation(\boldsymbol{y})} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}.$$

Recall, we define covariance and standard deviation as follows:

$$\begin{aligned}
\sigma_{xy} &= \tfrac{1}{n-1} \textstyle\sum_{k=1}^{n} (x_k - \bar{x})(y_k - \bar{y}) \\
\sigma_x &= \sqrt{\tfrac{1}{n-1} \textstyle\sum_{k=1}^{n} (x_k - \bar{x})^2} \\
\sigma_y &= \sqrt{\tfrac{1}{n-1} \textstyle\sum_{k=1}^{n} (y_k - \bar{y})^2} \\
\bar{x} &= \tfrac{1}{n} \textstyle\sum_{k=1}^{n} x_k \\
\bar{y} &= \tfrac{1}{n} \textstyle\sum_{k=1}^{n} y_k.
\end{aligned}$$

Example 2.31. *(Perfect Correlation) Correlation is always in the range -1 to 1. A correlation of 1 (resp. -1) means that $\boldsymbol{x}$, $\boldsymbol{y}$ have a perfect positive (resp. negative) linear relationship, i.e. $x_k = ay_k + b$, for some constants a, b. The following two vectors illustrate cases where the correlation is -1 and $+1$, respectively. In the first case, the means of $\boldsymbol{x}$, $\boldsymbol{y}$ were chosen to be 0 for simplicity.*

$$\begin{aligned}
\boldsymbol{x} &= \begin{pmatrix} -3 & 6 & 0 & 3 & -6 \end{pmatrix}^T \\
\boldsymbol{y} &= \begin{pmatrix} 1 & -2 & 0 & -1 & 2 \end{pmatrix}^T \\
corr(\boldsymbol{x}, \boldsymbol{y}) &= -1, \quad and \ x_k = -3y_k
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{x} &= \begin{pmatrix} 3 & 6 & 0 & 3 & 6 \end{pmatrix}^T \\
\boldsymbol{y} &= \begin{pmatrix} 1 & 2 & 0 & 1 & 2 \end{pmatrix}^T \\
corr(\boldsymbol{x}, \boldsymbol{y}) &= 1, \quad and \ x_k = 3y_k
\end{aligned}$$

Example 2.32. *(Nonlinear Relationships) If the correlation is 0, then there is no linear relationship between the two sets of values. However, nonlinear relationships can still exist. In the following example, $y_k = x_k^2$, but their correlation is 0:*

$$\begin{aligned}
\boldsymbol{x} &= \begin{pmatrix} -3 & -2 & -1 & 0 & 1 & 2 & 3 \end{pmatrix}^T \\
\boldsymbol{y} &= \begin{pmatrix} 9 & 4 & 1 & 0 & 1 & 4 & 9 \end{pmatrix}^T.
\end{aligned}$$

Example 2.33. *(Visualising correlation) We can also judge the correlation between two vectors by plotting pairs of corresponding values of $\boldsymbol{x}$, $\boldsymbol{y}$ in a scatter plot. In Figure 2.17,*

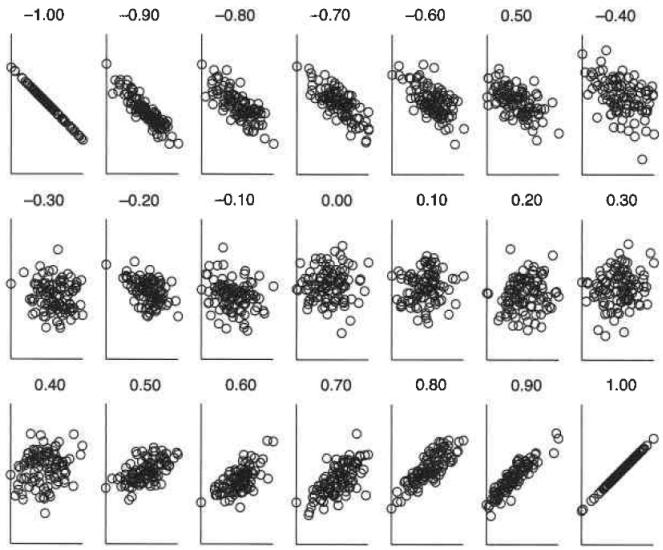


Figure 2.17: Scatter plots illustrating correlations from -1 to 1. [2]

we have a number of scatter plots for $\boldsymbol{x}$, $\boldsymbol{y}$ *consisting of a set of 30 pairs of values that are randomly generated (with a normal distribution) so that the correlation of* $\boldsymbol{x}$, $\boldsymbol{y}$ *ranges from* -1 *to 1. Each circle in a plot represents one of the 30 pairs of* $\boldsymbol{x}$, $\boldsymbol{y}$ *values; its x coordinate is the value of that pair for* $\boldsymbol{x}$*, while its y coordinate is the value of the same pair for* $\boldsymbol{y}$*.*

If we transform $\boldsymbol{x}$, $\boldsymbol{y}$ by subtracting off their means and then normalising so that their lengths are 1, then we can calculate their correlation by taking the scalar product. We denote these transformed vectors of $\boldsymbol{x}$, $\boldsymbol{y}$ by $\boldsymbol{x}'$, $\boldsymbol{y}'$, respectively. This highlights an interesting relationship between the correlation measure and the cosine measure. Specifically, the correlation between $\boldsymbol{x}$, $\boldsymbol{y}$ is identical to the cosine between $\boldsymbol{x}'$, $\boldsymbol{y}'$. However, the cosine between $\boldsymbol{x}$, $\boldsymbol{y}$ is not the same as the cosine between $\boldsymbol{x}'$, $\boldsymbol{y}'$, even though they both have the same correlation measure. In general, the correlation between two vectors is equal to the cosine measure only in the special case when the means of the two vectors are 0.

Part III

Data Mining and Postprocessing

Chapter 3

Classification

Further reading: Chapter 3 of [2]

Classification is a data mining task in which we attempt to extract models describing important data classes. Such models, called *classifiers*, predict categorical (discrete, unordered) class labels.

e.g. We can build a classification model to categorise bank loan applications as either safe or risky. This could include looking at attributes relating to an applicant's income, savings, financial history etc.

e.g. We can build a classification model to categorise lizards into one of several species. This could include looking at attributes relating to length, width, colour, diet, waking habits (nocturnal or diurnal) etc.

e.g. We can build a classification model that categorises email as spam or not spam. This could include looking at attributes relating to header and content etc.

e.g. We could classify cells as malignant or benign based upon MRI scans.

e.g. We could classify galaxies based upon their shapes. Are they a spiral galaxy or an elliptical galaxy, for example.

Such analysis can help provide a better understanding of the data at large. Many classification methods have been proposed by researchers in machine learning, pattern recognition, and statistics. Most algorithms typically assume a small data size but recent data mining research has built on such algorithms to introduce scalable classification and prediction techniques. While there are many different classification techniques, we are going to focus on a technique known as decision tree induction.

3.1 Preliminaries

The input data for a classification task is a collection of records. Each record (also known as an instance or example) is characterised by a tuple $(\mathbf{x}, y)$, where $\mathbf{x}$ is the attribute set and y is a special attribute, designated as the class label (also known as category or target attribute). Note that the attributes can be discrete or continuous (or a combination of both), but the class label must be a discrete attribute. This is a key characteristic that distinguishes classification from *regression*. For example, suppose we have a marketing manager who wants to guess whether a customer with a given profile will buy a new computer, say. This is a

classification task where the classifiers are 'yes' and 'no'. However, if the manager wanted instead to predict how much this given customer will spend, then this becomes one of *numeric prediction*, or regression. In this case, y is a continuous attribute and we need to employ methods, such as those seen in Appendix A. There are, of course, other methods of regression.

Example 3.1. *Consider the following table which shows a sample data set used for classifying vertebrates into one of the following categories: mammal, bird, fish, reptile, or amphibian.*

Name	*Body Temperature*	*Skin Cover*	*Gives Birth*	*Aquatic Creature*	*Aerial Creature*	*Has Legs*	*Hibernates*	*Class Label*
human	*warm-blooded*	*hair*	*yes*	*no*	*no*	*yes*	*no*	*mammal*
python	*cold-blooded*	*scales*	*no*	*no*	*no*	*no*	*yes*	*reptile*
salmon	*cold-blooded*	*scales*	*no*	*yes*	*no*	*no*	*no*	*fish*
whale	*warm-blooded*	*hair*	*yes*	*yes*	*no*	*no*	*no*	*mammal*
frog	*cold-blooded*	*none*	*no*	*semi*	*no*	*yes*	*yes*	*amphibian*
komodo dragon	*cold-blooded*	*scales*	*no*	*no*	*no*	*yes*	*no*	*reptile*
bat	*warm-blooded*	*hair*	*yes*	*no*	*yes*	*yes*	*yes*	*mammal*
pigeon	*warm-blooded*	*feathers*	*no*	*no*	*yes*	*yes*	*no*	*bird*
cat	*warm-blooded*	*fur*	*yes*	*no*	*no*	*yes*	*no*	*mammal*
leopard shark	*cold-blooded*	*scales*	*yes*	*yes*	*no*	*no*	*no*	*fish*
turtle	*cold-blooded*	*scales*	*no*	*semi*	*no*	*yes*	*no*	*reptile*
penguin	*warm-blooded*	*feathers*	*no*	*semi*	*no*	*yes*	*no*	*bird*
porcupine	*warm-blooded*	*quills*	*yes*	*no*	*no*	*yes*	*yes*	*mammal*
eel	*cold-blooded*	*scales*	*no*	*yes*	*no*	*no*	*no*	*fish*
salamander	*cold-blooded*	*none*	*no*	*semi*	*no*	*yes*	*yes*	*amphibian*

The idea is to construct a model based on this data, and then use this model to characterise a creature not present in the set. For example, suppose we encounter a gila monster which is cold-blooded, has skin covered in scales, does not give birth, is not aquatic, is not aerial, has legs and hibernates. What class label should we give it, based on the data set above?

Classification is the task of learning a *target function* f (also known informally as a *classification model* that maps each attribute set $\mathbf{x}$ to one of the predefined class labels y. Classification techniques are most suited for predicting or describing data sets with binary or nominal categories. They are less effective for ordinal categories (e.g. to classify a person as a member of a high-, medium, or low-income group) because they do not consider the implicit order among the categories. Other forms of relationships such as subclass/superclass relationships (e.g. humans and apes are primates, which in turn is a subclass of mammals) are also ignored.

To actually construct this function, we proceed in two steps.

General Approach to Classification:

1. Learning Step - this is where the classification model $y = f(\mathbf{x})$ is constructed.

   - Training Phase

     Here, we take a predetermined set of data classes or concepts. Our classification then uses a *learning agorithm* to build the classifier by analysing (or 'learning from') a *training set* made up of database tuples and their associated class labels. Each tuple is assumed to belong to a predefined class as determined by another database attribute called the *class label attribute.*

   - The individual tuples making up the training set are referred to as *training tuples* and are randomly sampled from the database being analysed. We sometimes also call these data tuples *samples, examples, instances, data points* or *objects.*

   - Supervised learning

     Because the class label of each training tuple is provided, this step is also known as *supervised learning* (i.e. the learning of the classifier is 'supervised' in that it is told to which class each training tuple belongs). This contrasts with *unsupervised learning* (or *clustering*), in which the class label of each training tuple is not known, and the number or set of classes to be learned may not be known in advance. In this case, we can use clustering to try to determine 'groups of similar tuples', and from there attempt to derive labels.

   - The class label attribute is discrete-valued and unordered. Our classification model is represented in the form of rules, decision trees or mathematical formulae. This could include neural networks, support vector machines, naive Bayes classifiers, and so on.

2. Classification Step - where the model $y = f(\mathbf{x})$ is used to predict class labels for the given data.

   - First, the predictive accuracy of the classifier is estimated. If we were to use the training set to measure the classifier's accuracy, this estimate would likely be optimistic, because the classifier tends to *overfit* the data (i.e. during learning it may incorporate some particular anomalies of the training data that are not present in the general data set overall). As such, our goal is to not only create a model that fits the data, but to fit the data *and* be generalisable.

- A *test* set is used, made up of *test tuples* and their associated class labels. They are independent of the training tuples, meaning that they were not used to construct the classifier.
- The accuracy of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier. We simply compare the associated class label of each test tuple compared with the learned classifier's class prediction for that tuple. If the accuracy of the classifier is considered acceptable, the classifier can be used to classify future data tuples, for which the class label is unknown.
- For a binary classification problem, we can represent the accuracy testing step using a *confusion matrix*:

		Predicted	
		Class = 1	Class = 0
Actual	Class = 1	f_{11}	f_{10}
	Class = 0	f_{01}	f_{00}

Each entry f_{ij} in this table denotes the number or records from class i predicted to be of class j. For instance, f_{01} is the number of records from class 0 incorrectly predicted to be of class 1. Based on the entries in the confusion matrix, the total number of correct predictions made by the model is $f_{11}+f_{00}$ and the total number of incorrect predictions if $f_{10}+f_{01}$. We can then compute accuracy as follows:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} = \frac{f_{11}+f_{00}}{f_{11}+f_{10}+f_{01}+f_{00}}.$$

Equivalently, the performance of a model can be expressed in terms of its *error rate*, which is given as follows:

$$Error\ rate = \frac{Number\ of\ wrong\ predictions}{Total\ number\ of\ predictions} = \frac{f_{10}+f_{01}}{f_{11}+f_{10}+f_{01}+f_{00}}.$$

Most classification algorithms seek models that attain the highest accuracy, or equivalently, the lowest error rate when applied to the test set.

We summarise the above in Figure 3.1.

3.2 Decision Tree Classifier

Decision tree classifiers are a method of classification by which we employ a flowchart-like tree structure, where each internal node denotes a test on an attribute, and each branch represents an outcome of the test, and each terminal node holds a class label.

As an example, consider the table shown in Example 3.1 and suppose we wish to distinguish mammals from non-mammals. Perhaps we have a new species and want to decide if it is a mammal. The first question we may ask is whether the species is cold- or warm-blooded.

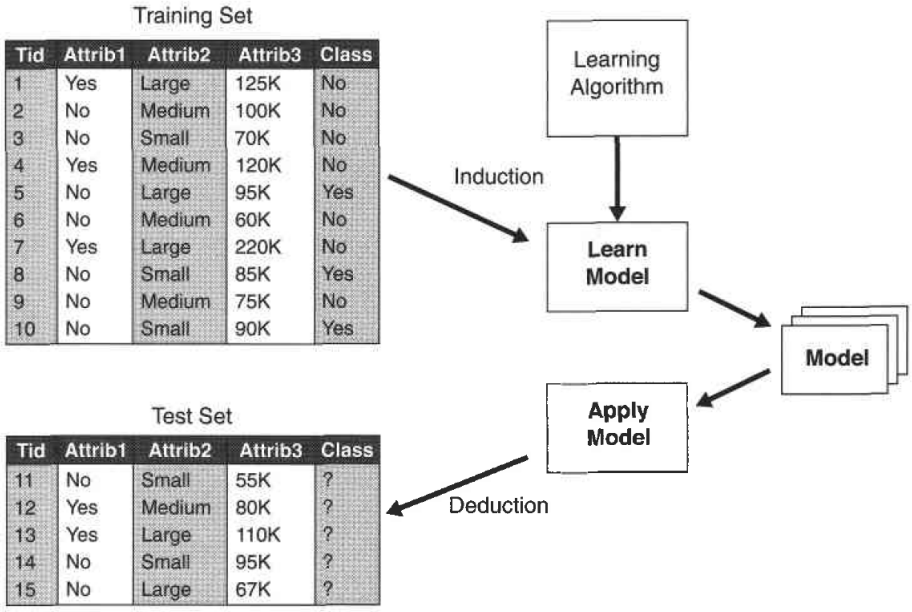


Figure 3.1: General approach for building a classification model. [2]

If it is cold-blooded, then it is definitely not a mammal. Otherwise, it is either a bird or a mammal. Suppose the species is warm-blooded. The next question we may ask is whether the females of the species gives birth to their young. Those that do are mammals, while those that do not are either birds, or egg-laying mammals such as the platypus or spiny anteater. We therefore ask if the species has mammary glands. If they do, then they are definitely mammals. Otherwise, they are birds. See Figure 3.2 for an example of this in action.

In this way we construct a decision tree with three types of nodes:

- Root node

  These have no incoming links and zero or more outgoing links. They are the topmost nodes and are linked with the first attribute test condition.

- Branch/Internal node

  These have exactly one incoming link and two or more outgoing links.

- Leaf/Terminal node

  These have exactly one incoming link and no outgoing links, and are associated with a class label. They are linked with an attribute test condition.

We label the various nodes for the above example in Figure 3.3.

Of course, our intention is always to find the optimal decision tree, but due to the exponential size of the search space, this is often a computationally expensive task. As such, efficient algorithms have been developed to induce a reasonably accurate decision (if not

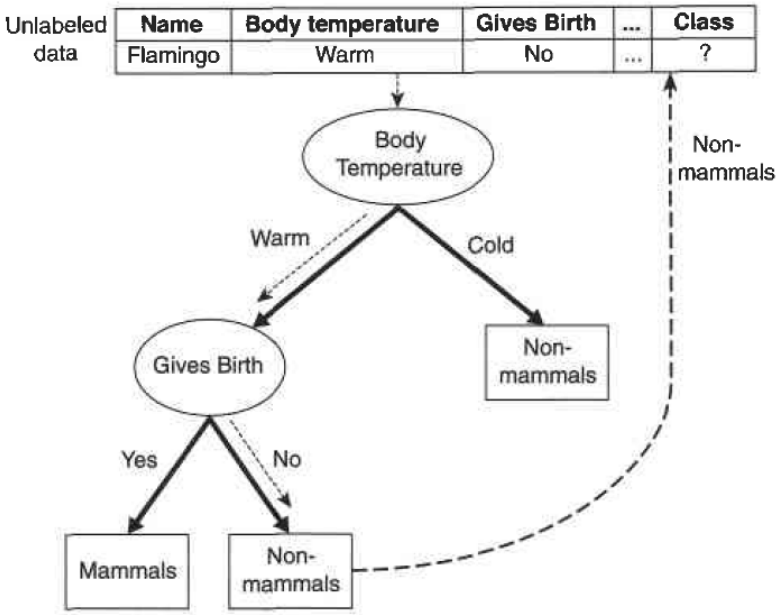


Figure 3.2: Classifying an unlabelled vertebrate. The dashed lines represent the outcomes of applying various attribute test conditions on the unlabelled vertebrate. The vertebrate is eventually assigned the `Non-mammal` class. [2]

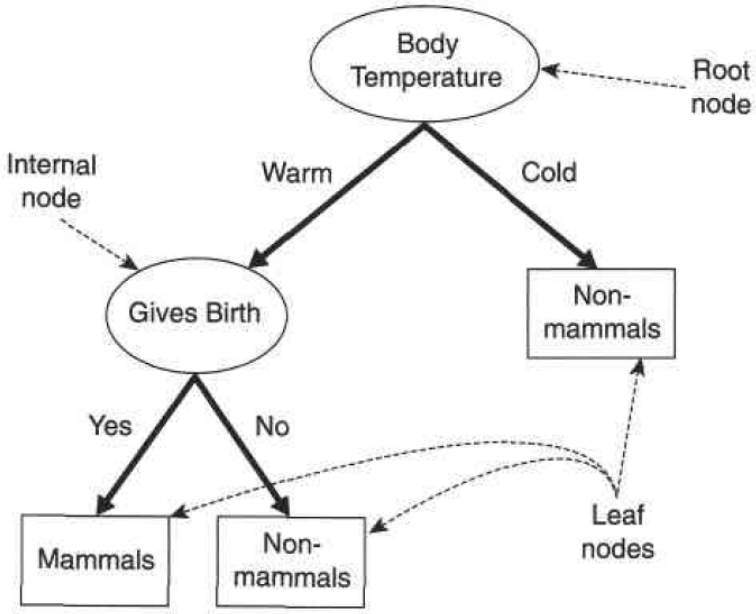


Figure 3.3: A decision tree for the mammal classification problem, with nodes. [2]

optimal) in a reasonable amount of time. Such algorithms usually employ a greedy strategy, growing the decision tree in a top-down fashion.

N.B. When decision trees are built, many of the branches may reflect noise or outliers in the training data. *Tree pruning* attempts to identify and remove such branches, with the goal of improving classification accuracy on unseen data.

An early example of the aforementioned algorithms is the so-called *Hunt algorithm*, named after E.B. Hunt. Hunts work, along with that of J. Marin and P.T. Stone lay the foundations for many current implementations of decision tree classifiers. These include algorithms developed by J. Ross Quinlan, namely ID3 (Iterative Dichotomoiser) and C4.5 (a successor of ID3), as well as the CART (Classification And Regression Trees) algorithm which was developed by L. Breiman, J. Friedman, R. Olshen and C. Stone. ID3 and CART were invented independently of one another at around the same time (the 1980s), yet follow a similar approach for learning decision trees from training tuples. Both adopt a greedy approach in which decision trees are constructed in a top-down recursive divide-and-conquer manner. Most algorithms for decision tree classification also follow a top-down approach. They start with a training set of tuples and their associated class labels and the training set is then recursively partitioned into smaller subsets as the tree is being build. The rough idea is that a child leaf node is created for each outcome of an attribute test condition and the instances associated with the parent node are distributed to the children based on the rest outcomes. This step is then recursively applied to each child nose, so long as it has labels of more than one class. If all the instances associated with a leaf node have identical class labels, then the node is not expanded any further. A basic algorithm is given in Figure 3.4, whose strategy is as follows:

1. We call the algorithm with three parameters: D, `attribute_list` and `Attribute_selection_method`.
    - We refer to D as a data partition. Initially, it is the complete set of training tuples and their associated class labels.
    - The parameter `attribute_list` is a list of attributes describing the tuples.
    - `Attribute_selection_method` specifies a heuristic procedure for selecting the attribute that 'best' discriminates the given tuples according to class. This procedure employs an attribute selection measure such as information gain or the Gini index (see later).
        - Whether the tree is strictly binary (each node has at most two children) is generally driven by the attribute selection measure.
        - Some attribute measures, such as the Gini index, enforce the resulting tree to be binary. Others, like information gain, do not.
2. The tree starts as a single node, N, representing the training tuples in D (Step 1).[1]

[1]The partition of class-labelled training tuples at node N is the set of tuples that follow a path from the root of the tree to node N when being processed by the tree.

3. If the tuples in D are all of the same class, then node N becomes a leaf and is labelled with that class (Steps 2 and 3). Note that Steps 4 and 5 are terminating conditions. All terminating conditions are explained at the end of the algorithm.

4. Otherwise, the algorithm calls `attribute_selection_method` to determine the *splitting criterion*.

   - The splitting criterion tells us which attribute to test at node N by determining the 'best' way to separate or partition the tuples in D into individual classes (Step 6).
   - The splitting criterion also tells us which branches to grow from node N with respect to the outcomes of the chosen test. More specifically, the splitting criterion indicates the *splitting attribute* and may also indicate either a *split-point* or a *splitting subset*.
   - The splitting criterion is determined so that, ideally, the resulting partitions at each branch are as 'pure' as possible.
     - A partition is *pure* if all the tuples in it belong to the same class.

     In other words, if we split up the tuples in D according to the mutually exclusive outcomes of the splitting criterion, we hope for the resulting partitions to be as pure as possible.

5. The node N is labelled with the splitting criterion, which serves as a test at the node (Step 7). A branch is grown from node N for each of the outcomes of the splitting criterion. The tuples in D are partitioned accordingly (Steps 10 and 11). There are then three possible scenarios. Let A be the splitting attribute which has ν distinct values, $\{a_1, a_2, \ldots, a_\nu\}$, based on the training data, (see Figure 3.5).

   (a) A is discrete valued

   In this case, the outcomes of the test at node N correspond directly to the known values of A. A branch is created for each known value, a_j, of A and labelled with that figure (see Figure 3.5(a)). Partition D_j is the subset of class-labelled tuples in D having value a_j of A. Because all the tuples in a given partition have the same value for A, A need not be considered in any future partitioning of the tuples. Therefore, it is removed from `attribute_list` (Steps 8 and 0.

   (b) A is continuous-valued

   In this case, the test at node N has two possible outcomes, corresponding to the condition $A \leq split\_point$ and $A > split\_point$, respectively, where $split\_point$ is the split-point returned by `Attribute_selection_method` as part of the splitting criterion.

   In practice, a is often taken as the midpoint of two known adjacent values of A and therefore may not actually be a preexisting value of A from the training data.

   Two branches are grown from N and labelled according to the previous outcomes (see Figure 3.5(b)). The typles are partitioned such that D_1 holds the subset of class-labelled tuples in D for which $A \leq split\_point$, while D_2 holds the rest.

(c) A is discrete valued and a binary tree must be produced (as dictated by the attribute selection measure or algorithm being used)

The test at node N is of the form '$A \in S_A$?,' where S_A is the splitting subset for A, returned by `Attribute_selection_method` as part of the splitting criterion. It is a subset of the known values of A. If a given tuple has value a_j of A and if $a_j \in S_A$, then the test at node N is satisfied. Two branches are grown from N (see Figure 3.5(c)). By convention, the left branch out of N is labelled *yes* so that D_1 corresponds to the subset of class-labelled tuples in D that satisfy the test. The right branch out of N is labelled *no* so that D_2 corresponds to the subset of class-labelled tuples from D that do not satisfy the test.

6. The algorithm uses the same process recursively to form a decision tree for the tuples at each resulting partition, D_j of D (Step 14).

7. The recursive partitioning stops only when any one of the following terminating conditions is true:

   (a) All the tuples in partition D (represented at node N) belong to the same class (Steps 2 and 3).

   (b) There are no remaining attributes on which the tuples may be further partitioned (Step 4). In this case, *majority voting* is employed (Step 5). This involves converting node N into a leaf and labelling it with the most common class in D. Alternatively, the class distribution of the node tuples may be stored.

   (c) There are no tuples for a given branch; that is, a partition D_j is empty (Step 12). In this case, a leaf is created with the majority class in D (Step 13).

The computational complexity of the algorithm given training set D is $O(n \times |D| \times \log(|D|))$, where n is the number of attributes describing the tuples in D and $|D|$ is the number of training tuples in D. This means that the computational cost of growing a tree grows at most $n \times |D| \times \log(|D|)$ with $|D|$ tuples.

Example 3.2. *To illustrate the above algorithm, suppose we have the following training set:*

ID	*Home Owner*	*Marital Status*	*Annual Income*	*Defaulted Borrower*
1	*Yes*	*Single*	*125k*	*No*
2	*No*	*Married*	*100k*	*No*
3	*No*	*Single*	*70k*	*No*
4	*Yes*	*Married*	*120k*	*No*
5	*No*	*Divorced*	*95k*	*Yes*
6	*No*	*Married*	*60k*	*No*
7	*Yes*	*Divorced*	*220k*	*No*
8	*No*	*Single*	*85k*	*Yes*
9	*No*	*Married*	*75k*	*No*
10	*No*	*Single*	*90k*	*Yes*

The decision tree is then represented in Figure 3.6.

Algorithm: `Generate_decision_tree`

Generate a decision tree from the training tuples of data partition, D.

Input:

- Data partition, D, which is a set of training tuples and their associated class labels;
- `attribute_list`, the set of candidate attributes;
- `Attribute_selection_method`, a procedure to determine the splitting criterion that 'best' partitions the data tuples into individual classes. This criterion consists of a `splitting_attribute` and, possibly, either a *split point* or *splitting subset*.

Output: A decision tree.

Method:

(1) create a node N;
(2) **if** tuples in D are all of the same class, C, **then**
(3) return N as a leaf node labeled with the class C;
(4) **if** *attribute_list* is empty **then**
(5) return N as a leaf node labeled with the majority class in D; // majority voting
(6) apply **Attribute_selection_method**(D, *attribute_list*) to **find** the "best" *splitting_criterion*;
(7) label node N with *splitting_criterion*;
(8) **if** *splitting_attribute* is discrete-valued **and**
multiway splits allowed **then** // not restricted to binary trees
(9) *attribute_list* ← *attribute_list* − *splitting_attribute*; // remove *splitting_attribute*
(10) **for each** outcome j of *splitting_criterion*
// partition the tuples and grow subtrees for each partition
(11) let D_j be the set of data tuples in D satisfying outcome j; // a partition
(12) **if** D_j is empty **then**
(13) attach a leaf labeled with the majority class in D to node N;
(14) **else** attach the node returned by **Generate_decision_tree**(D_j, *attribute_list*) to node N;
endfor
(15) return N;

Figure 3.4: Basic algorithm.

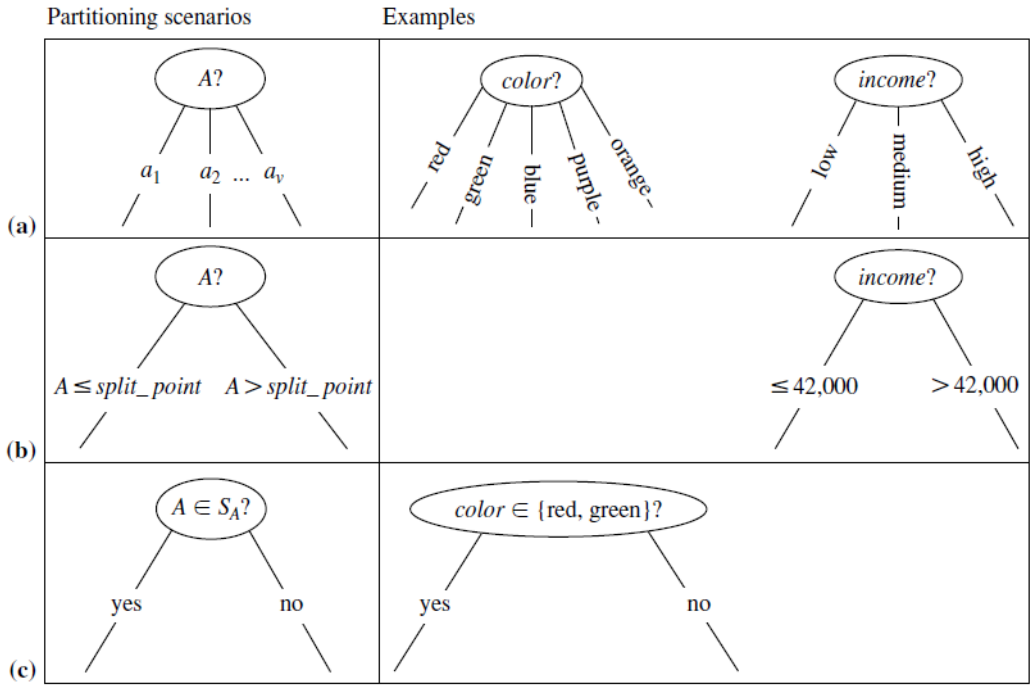


Figure 3.5: Splitting options for a decision tree. [2]

To work through this, we start at the single leaf node in the beginning. This node is labelled `Defaulted = No` *since the majority of borrowers did not default on their loan payments. The training error of this tree is 30% as three out of ten training instances have the class label* `Defaulted = Yes`*. The leaf node than therefore be expanded further as it contains instances from more than one class.*

Let `Home Owner` *be the attribute chosen to split the training instances. The resulting binary split on the* `Home Owner` *attribute is then shown in Figure 3.6(b). All the training instances for which* `Home Owner = Yes` *are propagated to the left child, and the rest to the right child. Hunt's algorithm is then recursively applied to each child. The left child becomes a leaf node labelled* `Defaulted = No` *since all instances associated with this node have identical class label* `Defaulted = No`*. The right child has instances from each label and so we need to further split this node. The resulting subtrees after recursively expanding the right child are shown in Figures 3.6(c)-(d).*

N.B. It is possible for all training instances associated with a node to have identical attribute values but different class labels. In this case, it is not possible to expand this node any further. One way to handle this is to declare it a leaf node and assign it the class label that occurs most frequently in the training instances associated with this node.

3.2.1 Attribute Selection Measures

There are several measures that can be used to determine the suitability of an attribute test condition. These measures try to give preference to attribute test conditions that partition

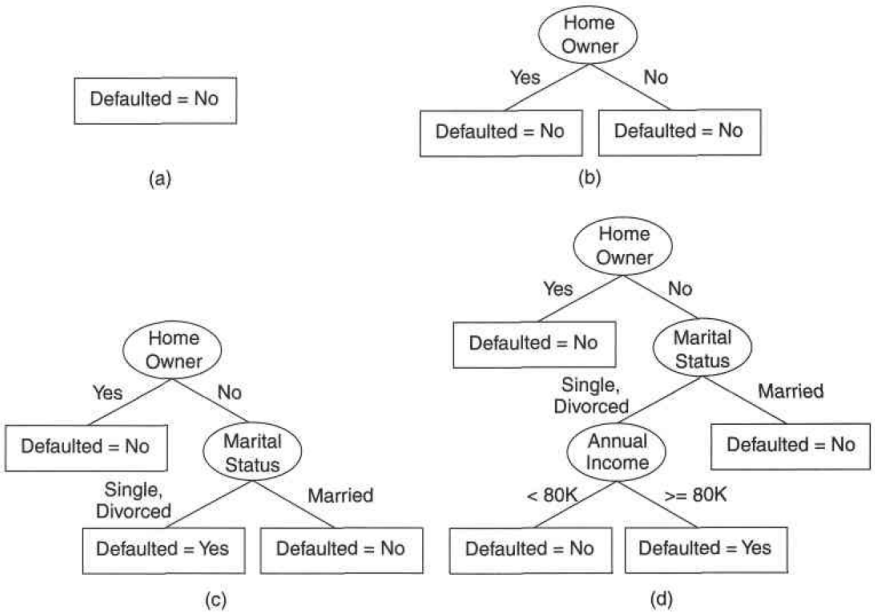


Figure 3.6: Hunt's algorithm for inducing decision trees. [2]

the training instances into *purer* subsets in the child nodes, which mostly have the same class labels. In general, larger trees are less desirable as they are more susceptible to model overfitting, a condition that may degrade the classification performance on unseen instances. They are also difficult to interpret and incur more training and test time compared to smaller trees. As such, the goal is to create a decision tree that is smaller, as opposed to larger. A big part of this is avoiding impure nodes containing training instances from multiple classes. These are likely to require several levels of node expansions, thereby increasing the depth of the tree.

- Impurity measure for a single node

The impurity of a node measures how dissimilar the class labels are for the data instances belonging to a common node. We can use several measures to evaluate the impurity of a node t, for instance:

$$\begin{aligned} Entropy = & \quad -\sum_{i=0}^{c-1} p_i(t) \log_2(p_i(t)), \\ Gini\ index = & \quad 1 - \sum_{i=0}^{c-1} (p_i(t))^2, \\ Classification\ error = & \quad 1 - max_i[p_i(t)], \end{aligned}$$

where $p_i(t)$ is the relative frequency of training instances that belong to class i at node t, c is the total number of classes, and $0 \log_2(0) = 0$ in entropy calculations. All three measures give a zero impurity value if a node contains instances from a single class and maximum impurity if the node has equal proportion of instances from multiple classes. We can compare the relative magnitudes of the impurity measures when applied to binary classification problems in Figure 3.7

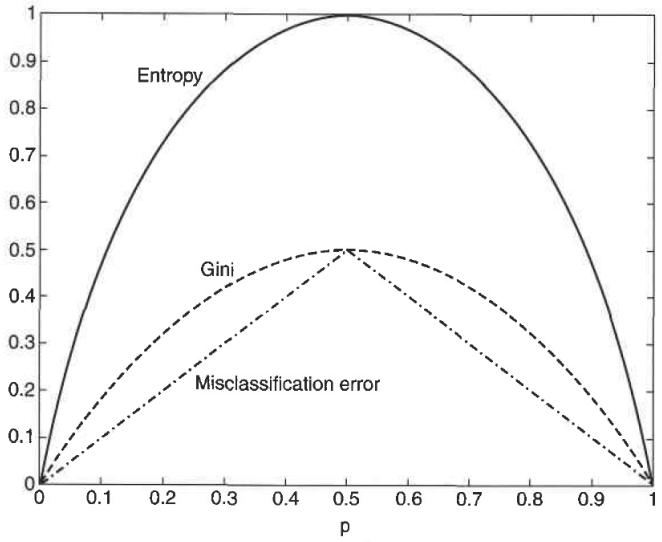


Figure 3.7: Comparison among the impurity measures for binary classification problems. [2]

In binary classification problems there are only two classes and so $p_0(t) + p_1(t) = 1$. The horizontal axis of Figure 3.7 refers to the fraction of instances that belong to one of the two classes. We note that in each case, the measures attain their maximum value when the class distribution is uniform, i.e. $p_0(t) = p_1(t) = 0.5$. Likewise, they obtain their minimum when all instances belong to a single class, i.e. $p_0(t)$ or $p_1(t)$ is equal to 1.

Example 3.3. *At node 1, we suppose* `Class = 0` *has a count of 0, while* `Class = 1` *has a count of 6. Then,*

$$\begin{aligned} Entropy = & \; -(0/6)\log_2(0/6) - (6/6)log_2(6/6) = 0 \\ Gini = & \; 1 - (0/6)^2 - (6/6)^2 = 0 \\ Error = & \; 1 - max[0/6,\, 6/6] = 0. \end{aligned}$$

At node 2, we suppose `Class = 0` *has a count of 1, while* `Class = 1` *has a count of 5. Then,*

$$\begin{aligned} Entropy = & \; -(1/6)\log_2(1/6) - (5/6)log_2(5/6) = 0.650 \\ Gini = & \; 1 - (1/6)^2 - (5/6)^2 = 0.278 \\ Error = & \; 1 - max[1/6,\, 5/6] = 0.167. \end{aligned}$$

At node 3, we suppose `Class = 0` *has a count of 3, while* `Class = 1` *has a count of 3. Then,*

$$\begin{aligned} Entropy = & \; -(3/6)\log_2(3/6) - (3/6)log_2(3/6) = 1 \\ Gini = & \; 1 - (3/6)^2 - (3/6)^2 = 0.5 \\ Error = & \; 1 - max[3/6,\, 3/6] = 0.5. \end{aligned}$$

Based on these computations, node N_1 has the lowest impurity value, followed by N_2 and then N_3.

As the above example demonstrates, there is a consistency among impurity measures, i.e. if a node $N-1$ has lower entropy than node N_2, then the Gini index and error rate of N_1 will also be lower than that of N_2. Nevertheless, the attribute chosen as a splitting criterion by the impurity measure can still be different.

- Collective Impurity of Child Nodes

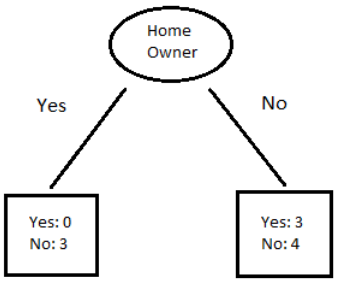
Consider now an attribute test condition that splits a node containing N training instances into k children, $\{v_1, v_2, \ldots, v_k\}$. Each child node represents a partition of the data resulting from one of the k outcomes of the attribute test condition. Further, we make the following definitions:

- Let $N(v_j)$ be the number of training instances associated with a child node v_j.
- Let the impurity value of v_j be $I(v_j)$.

Since a training instance in the parent node reaches node v_j for a fraction of $N(v_j)/N$ times, the collective impurity of the child nodes can be computed by taking a weighted sum of the impurities of the child nodes. This is given as follows:

$$I(children) = \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j).$$

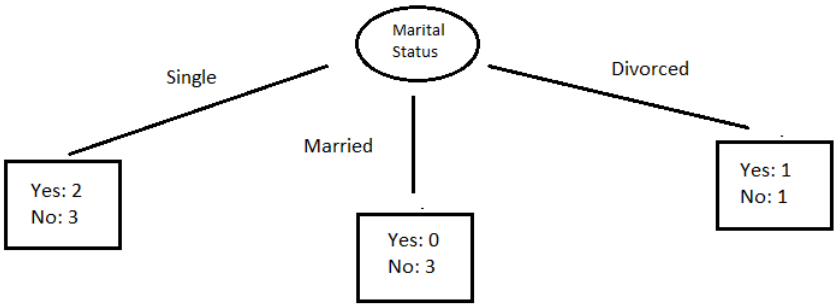
Example 3.4. *Return to our loan borrower classification problem. First, we can split the* `Home Owner` *attribute, generating two child nodes as follows:*



We can then compute the weighted entropy as follows:

$$\begin{aligned} I(\texttt{Home Owner = Yes}) &= -\tfrac{0}{3}\log_2(\tfrac{0}{3}) - \tfrac{3}{3}\log_2(\tfrac{3}{3}) = 0 \\ I(\texttt{Home Owner = No}) &= -\tfrac{3}{7}\log_2(\tfrac{3}{7}) - \tfrac{4}{7}\log_2(\tfrac{4}{7}) = 0.985 \\ I(\texttt{Home Owner}) &= \tfrac{3}{10} \times 0 + \tfrac{7}{10} \times 0.985 = 0.690. \end{aligned}$$

Alternatively, we can split on `Marital Status`*, as follows:*



This leads to three child nodes with weighted entropy given by

$$\begin{aligned} I(\texttt{Marital Status = Single}) &= -\tfrac{2}{5}\log_2(\tfrac{2}{5}) - \tfrac{3}{5}\log_2(\tfrac{3}{5}) = 0.971 \\ I(\texttt{Marital Status = Married}) &= -\tfrac{0}{3}\log_2(\tfrac{0}{3}) - \tfrac{3}{3}\log_2(\tfrac{3}{3}) = 0 \\ I(\texttt{Marital Status = Divorced}) &= -\tfrac{1}{2}\log_2(\tfrac{1}{2}) - \tfrac{1}{2}\log_2(\tfrac{1}{2}) = 1 \\ I(\texttt{Marital Status}) &= \tfrac{5}{10}\times 0.971 + \tfrac{3}{10}\times 0 + \tfrac{2}{10}\times 1 = 0.686. \end{aligned}$$

Thus, `Marital Status` *has a lower weighted entropy than* `Home Owner`*.*

To determine the suitability of an attribute test condition, we compare the degree of impurity of the parent node (before splitting) with the weighted degree of impurity of the child nodes (after splitting). The larger the difference, the better the test condition. We call this difference (denoted Δ) the *gain* in purity of an attribute test condition and we define it as follows:

$$\Delta = I(parent) - I(children),$$

where

- $I(parent)$ is the impurity of a node before splitting, and
- $I(children)$ is the weighted impurity after splitting.

When entropy is used as the impurity measure, the difference in entropy is commonly known as *information gain* and is denoted by Δ_{info}.

N.B. $\Delta \geq 0$ for any reasonable measure, such as those presented above.

Note also that the higher the gain, the purer are the classes in the child nodes relative to the parent node. Furthermore, maximising the gain at a given node is equivalent to minimising the weighted impurity measure of its children since $I(parent)$ is the same for all candidate attribute test conditions.

Example 3.5. *Consider the two candidate splits shown in Example 3.4 which involves the qualitative attributes* `Home Owner` *and* `Marital Status`*. The initial class distribution at the parent node is* (0.3, 0.7) *since there are three* `Yes` *and seven* `No` *classes in the training data. Thus,*

$$I(parent) = -\frac{3}{10}\log_2(\frac{3}{10}) - \frac{7}{10}\log_2(\frac{7}{10}) = 0.881.$$

The information gains for `Home Owner` *and* `Marital Status` *are each given by*

$$\Delta_{info}(\texttt{Home Owner}) = \quad 0.881 - 0.690 = 0.191$$
$$\Delta_{info}(\texttt{Marital Status}) = \quad 0.881 - 0.686 = 0.195.$$

We therefore conclude that the information gain for `Marital Status` *is higher (due to its lower weighted entropy), and will therefore be considered for splitting.*

Example 3.6. *Sticking with the loan borrower example, this time we consider building a decision tree using only binary splits and the Gini index as the impurity measure. We consider the following four candidate splitting criteria for the* `Home Owner` *and* `Marital Status` *attributes:*

1. `Home Owner`*:* `Yes` *or* `No`
2. `Marital Status`*:* `Single` *or* `Married/Divorced`
3. `Marital Status`*:* `Single/Married` *or* `Divorced`
4. `Marital Status`*:* `Single/Divorced` *or* `Married`

Using the data from Example 3.2, we have the following tables:

(1)	N_1	N_2
No	3	4
Yes	0	3
Gini	0.343	

(2)	N_1	N_2
No	3	4
Yes	2	1
Gini	0.400	

(3)	N_1	N_2
No	6	1
Yes	2	1
Gini	0.400	

(4)	N_1	N_2
No	4	3
Yes	3	0
Gini	0.343	

Since there are 3 borrowers in the training set who defaulted and 7 others who repaid their loan, the Gini index of the parent node before splitting is,

$$1 = \left(\frac{3}{10}\right)^2 - \left(\frac{7}{10}\right)^2 = 0.420.$$

If `Home Owner` *is chosen as the splitting attribute, the Gini index for the child nodes* N_1 *and* N_2 *is 0 and 0.490 respectively. The weighted average Gini index for the children is then given by,*

$$\frac{3}{10} \times 0 + \frac{7}{10} \times 0.490 = 0.343,$$

where the weights represent the proportion of training instances assigned to each child. The gain using `Home Owner` *as the splitting attribute is then given by*

$$\Delta_1 = 0.420 - 0.343 = 0.077.$$

We can likewise apply a binary split on the `Marital Status`*. However, since this is a nominal attribute with three outcomes, there are three possible ways to group the attribute values into a binary split. By repeating the arguments above, we obtain the gains given in the tables above. Based on these results,* `Home Owner` *and the last binary split using* `Marital Status` *are the best candidates since they both produce the lowest weighted average Gini index.*

N.B. Binary splits can also be used for ordinal attributes, so long as the binary partitioning of the attribute values does not violate the ordering property of the values.

Example 3.7. *This time, we consider binary splitting of quantitative attributes, such as that of* `Annual Income` *in the loan borrower dataset. In this case, we want to identify the best binary split* `Annual Income`$\leq \tau$*. Even though* τ *can take any value between the minimum and maximum values of annual income in the training set, it is sufficient to only consider the annual income values observed in the training set as candidate split positions. For each candidate* τ*, the training set is scanned once to count the number of borrowers with annual income less than or greater than* τ *along with their class proportions. We can then compute the Gini index at each candidate split position and choose the* τ *that produces the lowest value.*

Computing the Gini index at each candidate split position required $O(N)$ *operations, where* N *is the number of training instances. Since there are at most* N *possible candidates, the overall complexity of this brute-force approach is* $O(N^2)$*. It is possible to reduce the complexity of problem to* $O(N \log(N))$ *by using the following method.*

First, we sort the training instances based on their annual income, a one time cost that requires $O(N \log(N))$ *operations. The candidate split positions are given by the midpoints between every two adjacent sorted values: 55,000, 65,000, 75,000 and so on. For the first candidate, since none of the instances has an annual income less than or equal to 55,000, the Gini index for the child node with* `Annual Income`$< 55,000$ *is equal to zero. In contrast, there are 3 training instances of class* `Yes` *and 7 instances of class* `No` *with annual income greater than 55,000. The Gini index for this node is 0.420. The weighted average Gini index for the first candidate split position,* $\tau = 55,000$ *is equal to* $0 \times 0 + 1 \times 0.420 = 0.420$*.*

For the next candidate, $\tau = 65,000$ *and the class distribution of its child nodes can be obtained with a simple update of the distribution for the previous candidate. This is because, as* τ *increases from 55,000 to 65,000, there is only one training instance affected by the change. By examining the class label of the affected training instance, the new class distribution is obtained. For example, as* τ *increases to 65,000, there is only one borrower in the training set (with an annual income of 60,000) affected by this change. Since the class label for the borrower in* `No`*, the count for class* `No` *increases from 0 to 1 (for* `Annual Income`$\leq 65,000$*) and decreases from 7 to 6 (for* `Annual Income`$> 65,000$*). The distribution for the* `Yes` *class remains unaffected. The updated Gini index for this candidate split position is 0.400. We can see this in the following table [2]:*

Class	No		No		No		Yes		Yes		Yes		No		No		No		No	
	Annual Income																			
Sorted Values →	60		70		75		85		90		95		100		120		125		220	
Split Positions →	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		0.300		0.343		0.375		0.400		0.420	

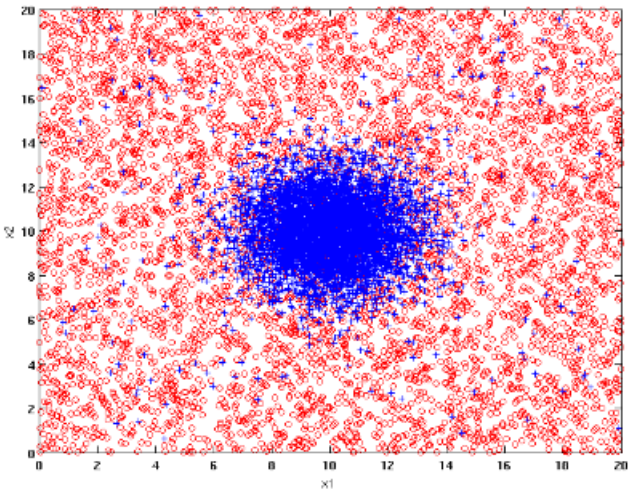
This procedure is repeated until the Gini index for all candidates are found. The best split position corresponds to the one that produces the lowest Gini index, which occurs at $\tau = 97500$. *Since the Gini index at each candidate split position can be computed in* $O(1)$ *time, the complexity of finding the best split position is* $O(N)$ *once all the values are kept sorted, a one-time operation that takes* $O(N \log(N))$ *time. The overall complexity of this method is thus* $O(N \log(N))$, *which is much smaller than the* $O(N^2)$ *time taken by the brute-force method. We can further reduce the amount of computation by only considering candidate split positions located between two adjacent sorted instances with different class labels. For example, we do not need to consider the candidate split positions located between 60,000 and 75,000 because all three instances with annual income in this range (60,000, 70,000 and 75,000) have the same class labels. Choosing a split position within this range only increases the degree of impurity, compared to a split position located outside this range. We can therefore ignore the split positions at* $\tau = 65,000$ *and* $\tau - 72,500$. *Likewise, we do not need to consider the split positions at 87,500, 92,500, 110,000, 122,500 and 172,500. This strategy reduces the number of candidate split positions to consider from 9 to just 2 (excluding the two boundary cases at* $\tau = 55,000$ *and* $\tau = 230,000$*).*

Exercise: Read pp. 154-156 of [2] to learn about the *gain ratio* and possible limitations of entropy and Gini index as impurity measures.

3.3 Model Overfitting

Thus far, our attempts have try to show the lowest error on the training set. However, in many cases a model may be an excellent fit for the training set, yet still show poor generalisation performance. We call this phenomenon, *model overfitting*. We demonstrate this with an example(due to [2]).

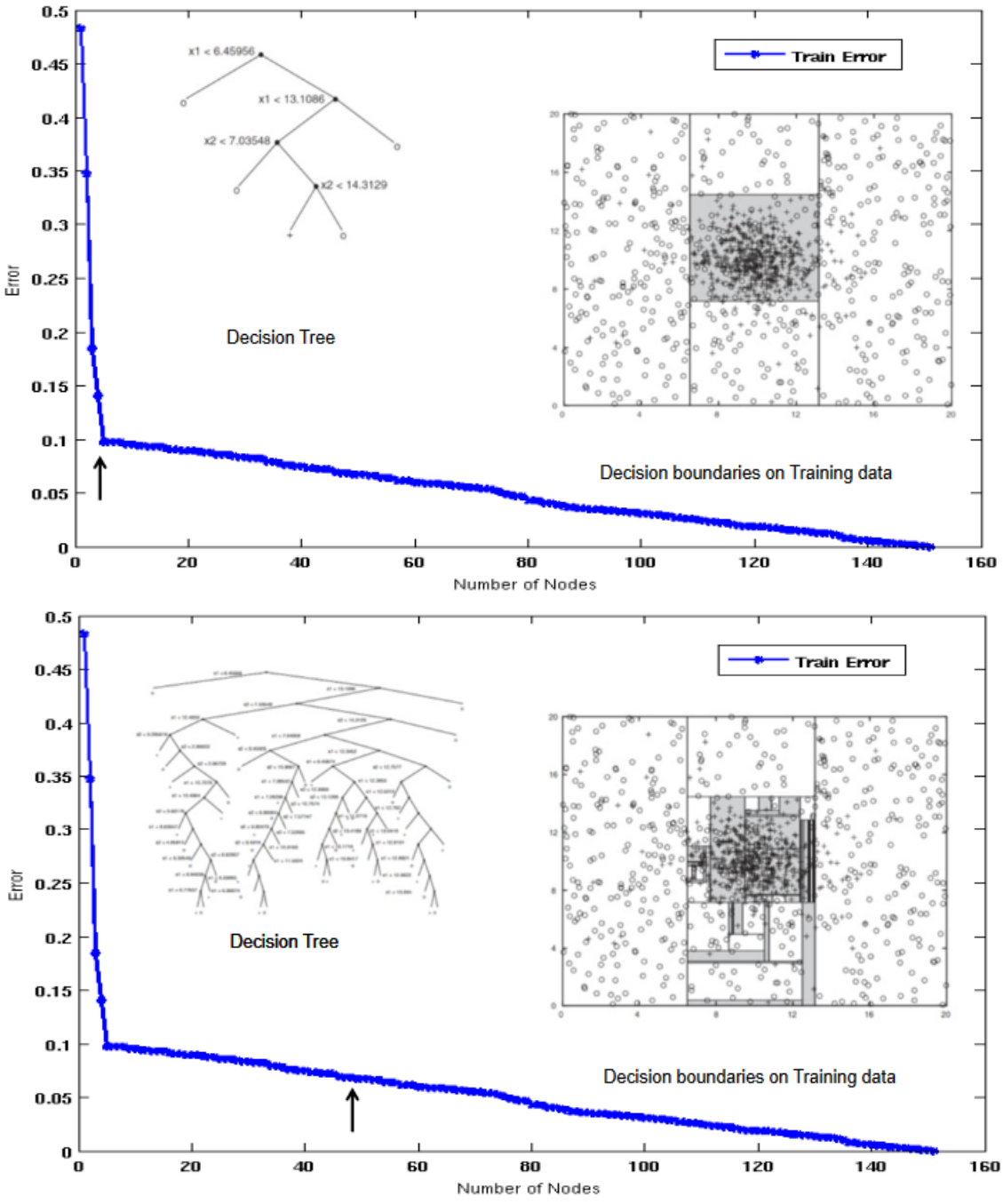
Consider the following two-dimensional data set in which we have two separate classes, represented as + and ∘, each of which has 5400 instances:



All instances belonging to the ∘ class were generated from a uniform distribution, while the + class is split. A Gaussian distribution centred at (10, 10) with unit variance was used to generate 5000 instances, with the remaining 400 instances sampled from the same uniform distribution as the ∘ class.

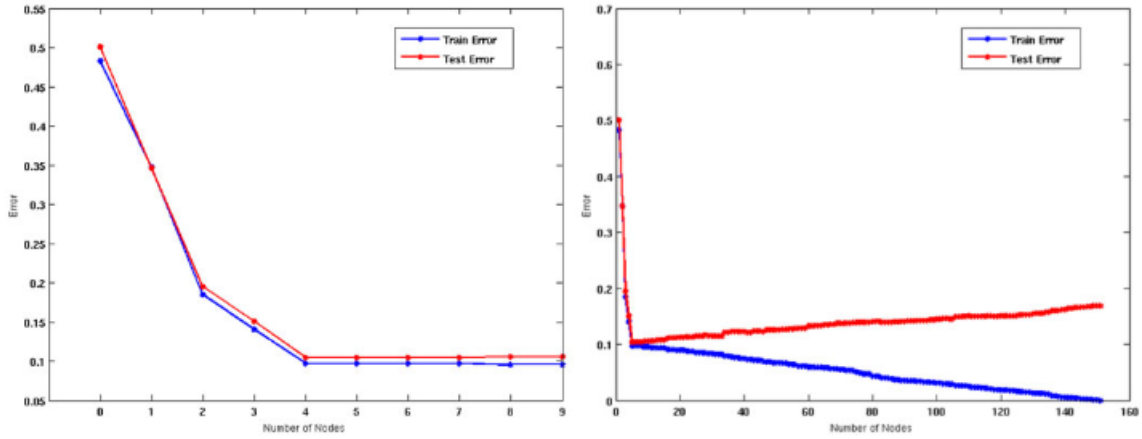
Visualising the above, it is clear that the + class can be mostly distinguished from the ∘ class by drawing a circle centred at (10, 10). To produce a classifer using this two-dimensional data set, we randomly sampled 10% of the data for training and used the remaining 90% to test. This training set can look quite representative of the whole data set and still cause problems. In any case, we use the Gini index as the impurity measure and construct decision trees of increasing sizes (number of leaf nodes), by recursively expanding a node into child nodes till every leaf node is pure. We can demonstrate this in the following pictures which show the case with 4 nodes and 50 nodes, respectively[2]:

[2]In the pictures, the shaded rectangles represent regions assigned to the + class.



Natural question: Which is better?

To answer this, we look to the following picture:



We note that at first, both error rates are large. This is the case when the tree has only one or two leaf nodes and so the decision tree is too simplistic to fully represent the true relationship between the attributes and the class labels. We call this situation *model underfitting*. As the tree size increases from 1 to 8 nodes, we observe two things:

1. **Both error rates decrease.** This reflects the idea that larger trees are able to represent more complex decision boundaries.

2. **The training and test error rates are quite close to each other.** This indicates that the performance on the training set is fairly representative of the generalisation performance.

However, as we increase the size of the tree from 8 to 150, something unusual happens. The training error continues to steadily decrease until it eventually reaches zero, but now the test error rate ceases to decrease any further beyond a certain tree size. Indeed, it starts to increase. In this way, the training error rate greatly underestimates the test error rate once the tree becomes too large. This behaviour, which may seem counter-intuitive at first, can be attributed to the phenomena of *model overfitting*. In the pursuit of minimising training error rate, we create an overly complex model that captures specific patterns in the training data but fails to learn the true nature of relationships between attributes and class labels in the overall data. This can be seen in the decision trees with 4 and 50 nodes. In the former, this fairly simple tree includes decision boundaries that provide a reasonable approximation to the ideal decision boundary, which in this case corresponds to a circle centred around the Gaussian distribution at (10, 10). The training and test error rates may still be non-zero, but they are very close to each other. This indicates that the patterns learned in the training set should generalise well over the test set. In contrast, the decision tree of size 50 is much more complex and has complicated decision boundaries. In trying to cover one or true + training instances, it has attempted to cover narrow regions in the input space. The prevalence of + instances in such regions is highly specific to the training set, as these regions are mostly dominated by ∘ instances in the overall data. Hence, in an attempt to perfectly fit the training data, the decision tree starts fine tuning itself to specific patterns in the training data, thereby leading to poor performance on an independently chosen test set.

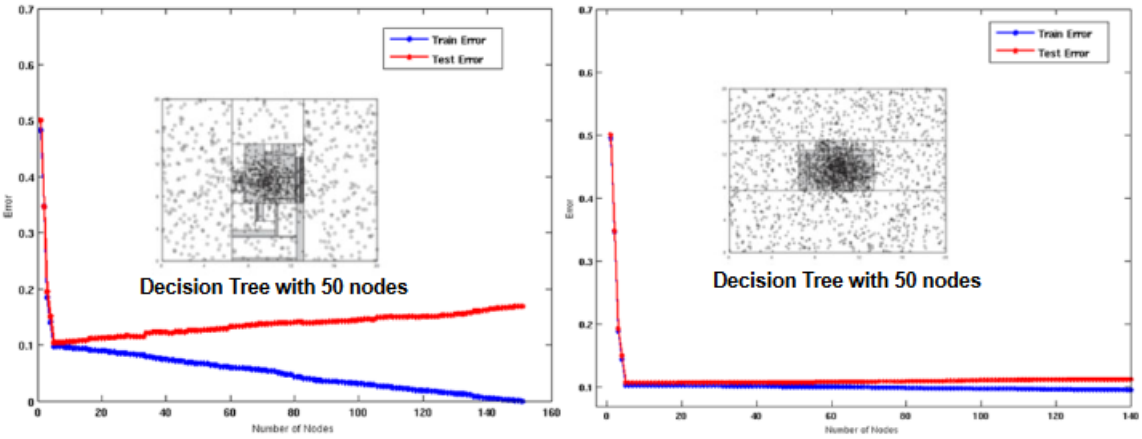
There are several factors that influence model overfitting. Three major factors include limited training size, high model complexity, and noise, which we now briefly discuss.

- Limited Training Size

A training set consisting of a finite number of instances can only ever provide a limited representation of the overall data. It is therefore possible that any patterns learned from a training set do not fully represent the true patterns in the overall data. This leads to model overfitting.

In general, increasing the size of a training set (number of training instances), increases the chance that the patterns learned from the training set resemble true patterns in the overall data. Hence, the effect of overfitting can be reduced by increasing the training data size.

Example 3.8. *Return to our previous example and suppose we use twice the number of training instances than what we had originally used in the experiments, i.e. we use 20% data for training instead of 10%. The rest is used for testing. Consider the following:*



This once again shows the training and test error rates as the size of the tree varies from 1 to 150, this time with twice the number of data instances. There are two major differences in the trends shown above from those seen earlier:

1. *Even though the training error rate decreases with increasing tree size in both cases, its rate of decrease is much smaller when using twice the training size.*

2. *For a given tree size, the gap between the training and test error rates is much smaller when we use twice the training size.*

These training differences suggest that the patterns learned using 20% of the data for training are more generalisable than those learned using 10% of data for training.

The above image also shows the decision boundaries for a decision tree with 50 nodes, learned using 20% of data for training. In contrast to the tree of the same size learned using 10% data for training, we can see that the decision tree is not capturing specific patterns of noisy + instances in the training set. Instead, the high model complexity of 50 leaf nodes is being effectively used to learn the boundaries of the + instances centred at $(10, 10)$.

- High Model Complexity

Generally, a more complex model has a greater ability to represent complex patterns in the data. Decision trees with a greater number of leaf nodes will in general be able to represent more complex decision boundaries, when compared to decision trees with fewer leaf nodes. However, an overly complex model also has a tendency to learn specific patterns in the training set that do not generalise well over unseen instances. Models with high complexity should therefore be judiciously used to avoid overfitting.

Question: How do we measure complexity?

One way is to look at the number of 'parameters' that need to be inferred from the training set. For example, in the case of decision tree induction, the attribute test conditions at internal nodes correspond to the parameters of the model that need to be inferred from the training set. A decision tree with a larger number of attribute test conditions (and consequently more leaf nodes) thus involves more 'parameters' and hence is more complex.

Suppose we have a class of models with a certain number of parameters. A learning algorithm tries to select the best combination of parameter values than maximises some evaluation metric (e.g. accuracy) over the training set. If the number of parameter value combinations (and hence the complexity) is large, then the algorithm has to select the best combination from a larger number of possibilities, using a limited training set. This leads to a greater chance of the algorithm picking a spurious combination. In this case, this combination maximises the evaluation but does so by random chance, and not because it has identified some pattern in the training data that generalises to the overall data. This is similar to the *multiple testing problem* in statistics.

Example 3.9 (Example of multiple testing problem)**.** *Consider the task of predicting whether the stock market will rise or fall in the next ten trading days. If a stock analyst makes random guesses the probability that their prediction is correct on any trading day is 0.5. However, the probability that they correctly predict at least nine out of ten times is extremely low:*

$$\frac{\binom{10}{9} + \binom{10}{10}}{2^{10}} = 0.0107.$$

Now suppose we wish to choose an investment advisor from a pool of 200 stock analysis. Our strategy is to select the analyst who makes the most number of correct predictions in the next ten trading days. The flaw in our thinking is that even if all the analysts make their predictions in a random fashion, the probability that at least one of them makes at least nine correct predictions is very high:

$$1 - (1 - 0.0107)^{200} = 0.8847.$$

Although each analyst has a low probability of predicting at least nine times correctly, considered together we have a high probability of finding at least one analyst who can do so. Of course, there is no guarantee in the future that such an analyst will continue to make accurate predictions by random guessing.

Relating this back to the problem of model overfitting, in the context of learning a classification model each combination of parameter values corresponds to an analyst, while the number of training instances corresponds to the number of days. Analogous to the task of selecting the best analyst who makes the most accurate predictions on consecutive days, the task of a learning algorithm is to select the best combination of parameters that results in the highest accuracy on the training set. If the number of parameter combinations is large but the training size is small, it is highly likely for the learning algorithm to choose a spurious parameter combination that provides high training accuracy by simply random chance.

- Presence of Noise

Consider the following training and test sets for a mammal classification problem. Note that two of the ten training records are mislabelled (bats and whales).

Training set for classifying mammals					
Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
porcupine	warm-blooded	yes	yes	yes	yes
cat	warm-blooded	yes	yes	no	yes
bat	warm-blooded	yes	no	yes	no∗
whale	warm-blooded	yes	no	no	no∗
salamander	cold-blooded	no	yes	yes	no
komodo dragon	cold-blooded	no	yes	no	no
python	cold-blooded	no	no	yes	no
salmon	cold-blooded	no	no	no	no
eagle	warm-blooded	no	no	no	no
guppy	cold-blooded	yes	no	no	no

Test set for classifying mammals					
Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
human	warm-blooded	yes	no	no	yes
pigeon	warm-blooded	no	no	no	no
elephant	warm-blooded	yes	yes	no	yes
leopard shark	cold-blooded	yes	no	no	no
turtle	cold-blooded	no	yes	no	no
penguin	warm-blooded	no	no	no	no
eel	cold-blooded	no	no	no	no
dolphin	warm-blooded	yes	no	no	yes
spiny anteater	warm-blooded	no	yes	yes	yes
gila monster	cold-blooded	no	yes	yes	no

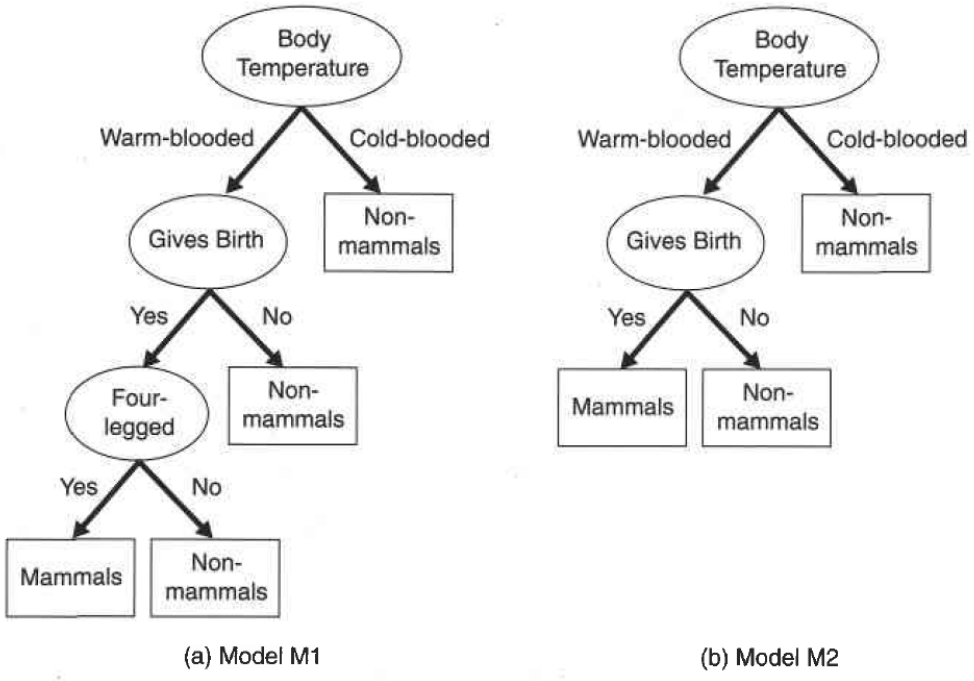


Figure 3.8: Decision tree induced from training set for classifying mammals. [2]

In Figure 3.8, there is a decision tree that perfectly fits the training data shown above (M1). Although the training error for the tree is zero, its error rate on the test is 30%. Both humans and dolphins were misclassified as non-mammals because their `Body Temperature`, `Gives Birth` and `Four-legged` are identical to the mislabelled records in the training set. Spiny anteaters, on the other hand, represent an exceptional case in which the class label of a test record contradicts the class labels of other similar records in the training set. Errors due to exceptional cases are often unavoidable and establish the minimum error rate achievable by any classifier.

In contrast, the decision tree M2 has a lower test error rate (10%) even though its training error rate is higher (20%). It is evident that the first decision tree M1 has overfitted the training data because there is a simpler model with lower error rate on the test set. The `Four-legged` attribute test condition in model M1 is spurious because it fits the mislabelled training records, which leads to misclassification in the test set.

3.4 Model Selection

Often, we have several possible classification models to choose from, many of which have varying levels of complexity. We want to choose the model that shows the lowest generalisation error rate. As we have seen, looking at test error rate alone is not enough to reliably choose a model. As such, we need to other means to influence our selection. In this section, we will see three generic approaches to estimate the generalisation performance of a model. We will then present specific strategies for using these approaches in the context of decision

tree induction.

1. Use a validation set

A popular strategy is to use a *validation set* which contains data not used for training the model. As this data is unseen by the model, error rates on a validation set are usually a better indicator of generalisation performance than the training error rate alone. As such, we want to find such *validation errors* and use these to influence our model selection.

To do so, we take our training set (call it $D.train$) and partition it into two smaller subsets (call them $D.tr$ and $D.val$). We then use $D.tr$ to train out model, and $D.val$ for the validation set. How we partition $D.train$ is up to us and is largely situation specific, e.g. we could use two-thirds for $D.tr$ and one-third for $D.val$. For any choice of classification model m that is trained on $D.tr$, we can then estimate its validation error rate on $D.val$, labelled $err_{val}(m)$. The model that shows the lowest value of $err_{val}(m)$ can then be selected as the preferred choice of model.

One limitation of this approach is its sensitivity to the sizes chosen from the partition. If we make $D.tr$ too small then this may result in the learning of a poor classification model since a smaller training set is likely to be less representative of the overall data. Likewise, if $D.val$ is too small, the validation error rate might not be representative of the generalisation error rate, thereby nullifying the point of this step.

Example 3.10. *Consider Figure 3.9 which shows two decision trees generated from the same training data (example due to [2]). The counts shown in the leaf nodes of the trees represent the class distribution of the training instances. We choose the labels based on which is more popular in the class distribution.*

Next, consider Figure 3.10 which shows the predicted labels at the leaf nodes of the decision trees. The counts beneath the leaf nodes represent the proportion of data instances in the validation set that reach each of the nodes. Based on our earlier notation, for T_L we have the following:

$$\begin{aligned} f_{11} &= 6 \\ f_{00} &= 4 \\ f_{10} &= 4 \\ f_{01} &= 2. \end{aligned}$$

Thus, the validation error rate for the left tree is $err_{val}(T_L) = \frac{6}{16} = 0.375$. Likewise, for T_R we have the following:

$$\begin{aligned} f_{11} &= 5 \\ f_{00} &= 4 \\ f_{10} &= 2 \\ f_{01} &= 5. \end{aligned}$$

Thus, the validation error rate for the right tree is $err_{val}(T_R) = \frac{7}{16} = 0.4375$. Based on this, the left tree is preferred over the right one.

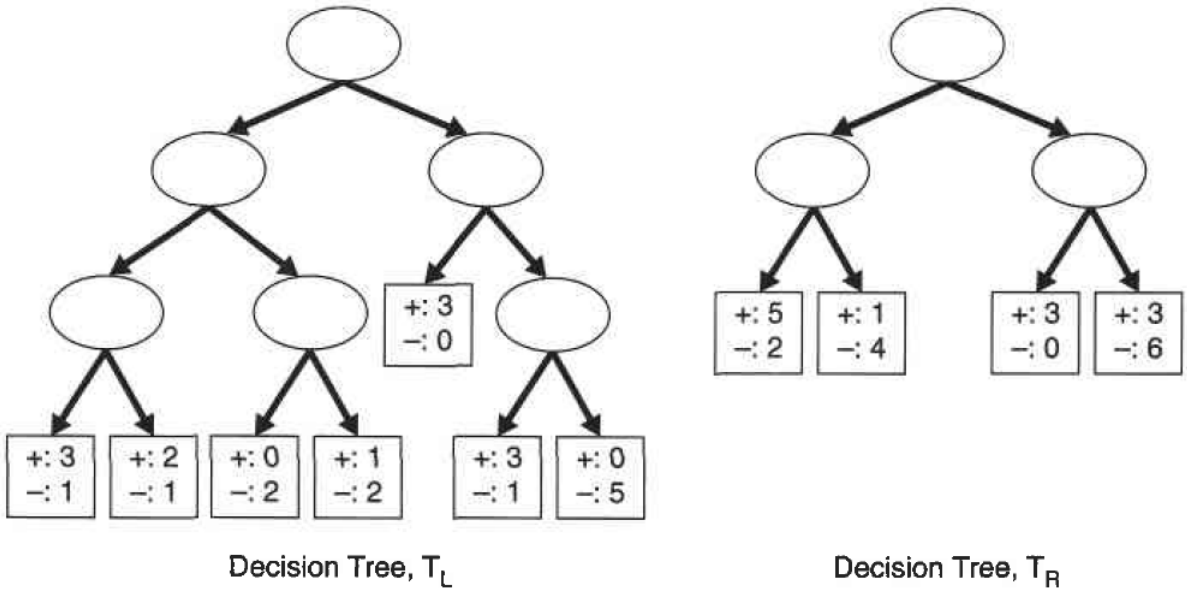


Figure 3.9: Two decision trees generated from the same training data. [2]

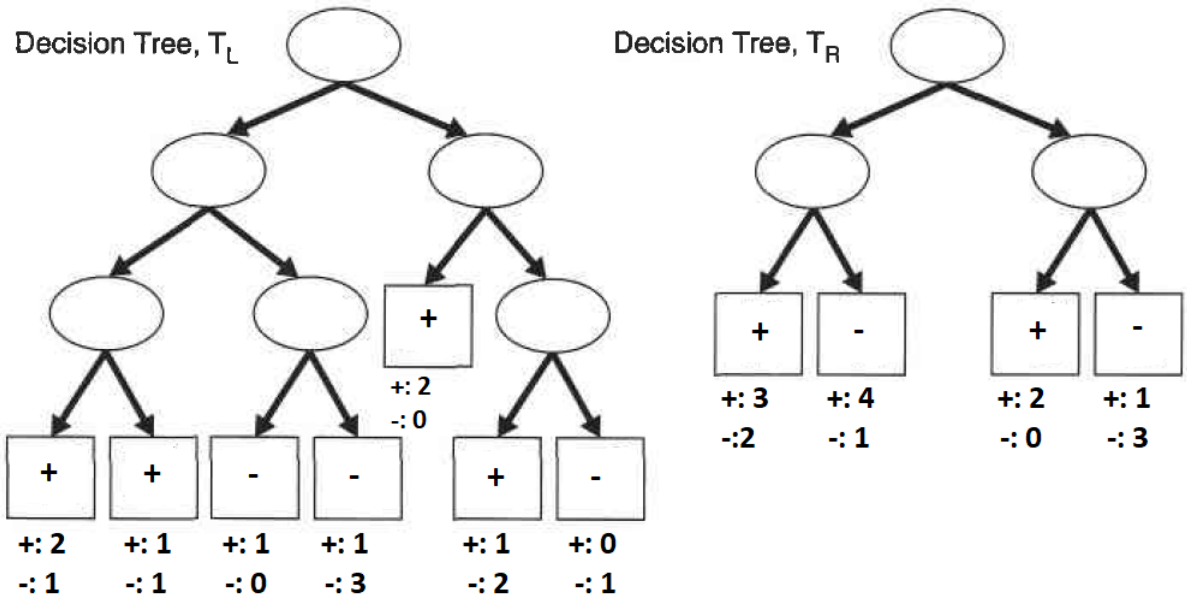


Figure 3.10: Class distribution of validation data.

2. Incorporating model complexity

As discussed previously, the chance for model overfitting increases as the model becomes more complex. As such, we need not only consider training error rate, but also model complexity. As a rough rule of thumb, we adhere to the Occam's razor principle, i.e. given two models with the same errors, the simpler model is preferred over the more complex model.

A generic approach to account for model complexity while estimated generalisation performance is formally described as follows. Suppose we have a training set $D.train$ and a class of models $\mathcal{M}$ (for example, $\mathcal{M}$ could represent the set of all possible decision trees). We consider a classification model m that is in $\mathcal{M}$, and are interested in estimating the generalisation error rate of m, denoted by $gen.error(m)$.

The training error rate of m ($train.error(m,\ D.train)$) can underestimate $gen.error(m)$ when the model complexity is high. We therefore represent $gen.error(m)$ as a function of the training error rate *and* model complexity of $\mathcal{M}$, denoted by $complexity(\mathcal{M})$. We define the function by,

$$gen.error(m) = train.error(m,\ D.train) + \alpha \times complexity(\mathcal{M}), \tag{3.11}$$

where α is a hyper-parameter that strikes a balance between minimising training error and reducing model complexity.

N.B. A higher value of α gives more emphasis to the model complexity in the estimation of generalisation performance.

To choose the right value of α we can make use of a validation set. For instance, we can iterate through a range of values of α and for every possible value we learn a model on a subset $D.tr$ of $D.train$. We then compute the validation error rate on a separate subset $D.val$ and select the value of α that provides the lowest validation error rate.

Equation 3.11 provides one possible approach for incorporating model complexity into the estimate of generalisation performance. This approach is at the heart of a number of techniques for estimating generalisation performance, such as:

- the structural risk minimisation principle;
- the Akaike's Information Criterion (AIC);
- the Bayesian Information Criterion (BIC).

In particular, the structural risk minimisation principle serves as the building block for learning support vector machines.

In the context of decision trees, the complexity of a decision tree can be estimated as the ratio of the number of leaf nodes to the number of training instances. Let k be the number of leaf nodes and N_{train} be the number of training instances. The complexity of a decision tree can then be described as $\frac{k}{N_{train}}$.

Intuition: For a larger training size, we can learn a decision tree with a larger number of leaf nodes without it becoming overly complex.

The generalisation error rate of a decision tree T can then be computed using Equation 3.11 as follows:

$$err_{gen}(T) = err(T) + \Omega \times \frac{k}{N_{train}},$$

where

- $err(T)$ is the training error of the decision tree, and
- Ω is a hyper-parameter that makes a trade-off between reducing training error and minimising model complexity, similar to α before.

We can view Ω as the relative cost of adding a leaf node relative to incurring a training error. In the literature, this is often referred to as the *pessimistic error estimate*, reflecting the idea that it assumes the generalisation error rate to be worse than the training error rate (by adding a penalty term for model complexity). In contrast, the *optimistic error estimate* (or *resubstitution estimate*) simply uses the training error as an estimate of the generalisation error rate.

Example 3.12. *Return to the decision tree in Figure 3.9. If each leaf node is labelled according to the majority class of training instances that reach the node, the training error rate for the left tree will be* $err(T_L) = \frac{4}{24} = 0.167$*, while the training error rate for the right tree will be* $err(T_R) = \frac{6}{24} = 0.25$*. Based on their error training rates alone,* T_L *would be preferred over* T_R*, even though* T_L *is more complex than* T_R*.*

Next, we assume that the cost associated with each leaf node is $\Omega = 0.5$*. Then the generalisation error estimate for* T_L *will be,*

$$err_{gen}(T_L) = \frac{4}{24} + 0.5 \times \frac{7}{24} = 0.3125$$

and the generalisation error estimate for T_R *will be*

$$err_{gen}(T_R) = \frac{6}{24} + 0.5 \times \frac{4}{24} = \frac{1}{3}.$$

Since T_L *has a lower generalisation error rate, it will still be preferred over* T_R*.*

Note that $\Omega = 0.5$ *implies that a node should always be expanded into its two child nodes if it improves the prediction of at least one training instance, since expanding a node is less costly than misclassifying a training instance. Alternatively, if* $\Omega = 1$*, then the generalisation error rate for* T_L *is* $err_{gen}(T_L) = \frac{11}{24} = 0.458$ *and for* T_R *is* $err_{gen}(T_R) = \frac{10}{24} = 0.417$*. In this case,* T_R *is the preferred option because it has a lower generalisation error rate.*

The above demonstrates the impact of choosing different values for Ω*. As with* α*, the value of* Ω *can be selected with the help of a validation set.*

3. Estimating statistical bounds

Instead of using Equation 3.11 to estimate the generalisation error rate of a model, we can instead apply a statistical correction to the training error rate of the model that is indicative of its model complexity. This can be done if either the probability distribution of training error is available, or can be assumed.

e.g. the number of errors committed by a leaf node in a decision tree can be assumed to follow a binomial distribution.

We can therefore compute an upper bound limit to the observed training error rate that can be used for model selection, as illustrated in the following example.

Example 3.13. *Consider the left-most branch of the binary decision trees shown in Figure 3.9. We note that the left-most lead node of T_R has been expanded into two child nodes in T_L. Before splitting, the training error rate of the node is $\frac{2}{7} = 0.286$. By approximating a binomial distribution with a normal distribution, the following upper bound of the training error rate e can be derived:*

$$e_{upper}(N,\, e,\, \alpha) = \frac{e + \frac{z^2_{\alpha/2}}{2N} + z_{\alpha/2}\sqrt{\frac{e(1-e)}{N} + \frac{z^2_{\alpha/2}}{4N^2}}}{1 + \frac{z^2_{\alpha/2}}{N}}, \tag{3.14}$$

where

- *α is the confidence level,*
- *$z_{\alpha/2}$ is the standardised value from a standard normal distribution, and*
- *N is the total number of training instances used to compute e.*

By replacing $\alpha = 24\%$, $N = 7$ and $e = \frac{2}{7}$, the upper bound for the error rate is $e_{upper}(7,\, \frac{2}{7},\, 0.25) = 0.503$, which corresponds to $7 \times 0.503 = 3.521$ errors.

If we now expand the node into its child nodes, as shown in T_L, then the training error rates for the child nodes are $\frac{1}{4} = 0.250$ and $\frac{1}{3} = 0.333$, respectively. Using Equation 3.14, the upper bounds of these error rates are $e_{upper}(4,\, \frac{1}{4},\, 0.25) = 0.537$ and $e_{upper}(3,\, \frac{1}{3},\, 0.25) = 0.650$, respectively. The overall training error of the child nodes is $4 \times 0.537 + 3 \times 0.650 = 4.098$, which is larger than the estimated error for the corresponding node in T_R, suggesting that it should not be split.

Building on these generic approaches, we now present two commonly used model selection strategies for decision tree induction. Both relate to the idea of tree pruning. To deal with problems of overfitting, tree pruning typically uses statistical measures to remove the least-reliable branches. Pruned trees tend to be smaller and less complex, hence tend to be easier to understand. They are also usually faster and better at correctly classifying independent test data than unpruned trees.

1. Prepruning (Early stopping rule)

   In this case, a tree is 'pruned' by halting its contruction early (e.g. by deciding not to further split or partition the subset of training tuples at a given node. To do this, a more restrictive stopping condition must be used (e.g. stop expanding a leaf node when the observed gain in the generalisation error estimate falls below a certain threshold. This estimate of the generalisation error can be computed using any of the approaches just presented (pessimistic error estimates, validation error estimates or using statistical bounds), and measures such as information gain, Gini index, and

so on, can be used to assess the 'goodness' of a split. There are difficulties, however, in choosing an appropriate threshold. High thresholds could result in oversimplified trees, whereas low thresholds could result in little simplification. Another drawback is that, even if no significant gain is obtained using one of the existing splitting criterion, subsequent splitting may result in better subtrees. Yet, such subtrees would not be reached if prepruning is used because of the greedy nature of decision tree induction.

2. Postpruning

   This is the more common approach and involves removing subtrees from a 'fully grown' tree. Trimming can be done by replaying a subtree with

   (a) a new leaf node whose class label is determined from the maority class of instances affiliated with the subtree (an approach known as *subtree replacement*, or

   (b) the most frequently used branch of the subtree (an approach known as *subtree raising*).

   The tree pruning step terminates when no further imporovement in the generalisation error estimate is observed beyond a certain threshold. Again, the estimates of generalisation error rate can be computed using any of the approaches presented above.

   Postpruning tends to give better results than prepruning because it makes pruning decisions based on a fully grown tree, unlike prepruning which can suffer from premature termination of the tree-growing process. However, for postpruning, the additional computations needed to grow the full tree may be wasted when the subtree is pruned.

See pp. 183-184 of [2] for an algorithm outlining the pruning process.

3.5 Model Evaluation

We complete this chapter by discussing ways to estimate a classification model's generalisation performance, i.e. its performance on unseen instances outside of $D.train$. Of course, in selecting a model (as we did in the last section), also computes an estimate of the generalisation performance using the training set $D.train$. However, these estimates are *biased* indicators of the performance on unseen instances since they were used to guide the selection of the classification model. For example, if we use the validation error rate for model section, the resulting model would be deliberately chosen to minimise the errors on the validation set. The validation error rate may thus under-estimate the true generalisation error rate, and hence cannot be reliably used for model evaluation.

A correct approach for model evaluation would therefore be to assess the performance of a learned model on a labelled test set that has not been used at any stage of model selection. This can be achieved by partitioning the entire set of labelled instanced D into two disjoint subsets, $D.train$ (used for model selection), and $D.test$ (used for computing the test error rate, err_{test}. We now present two different approaches for partitioning D into $D.train$ and $D.test$, and computing the test error rate, err_{test}.

- Holdout method

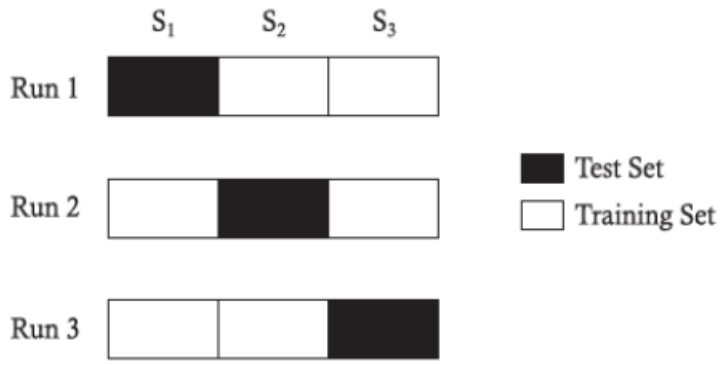


Figure 3.11: Example demonstrating the technique of 3-fold cross-validation. [2]

The most basic technique for partitioning a labelled data set is the *holdout method*. Here, the labelled set D is randomly partitioned into two disjoint sets, called the training set $D.train$ and the test set $D.test$. A classification model is then induced from $D.train$ using the model selection approaches already discussed, and its error rate on $D.test$, err_{test} is used as an estimate of the generalisation error rate.

The proportion of data reserved for training and for testing is generally situation specific, e.g. we could split them two-thirds and one-third, respectively. As with the trade-off faced when partitioning $D.train$ into $D.tr$ and $D.val$, choosing the right fraction of labelled data to be used for training/test is crucial and generally non-trivial. If the size of $D.train$ is small, the learned classification model may be improperly learned using an insufficient number of training instances. This results in a biased estimated of generalisation performance. Likewise, if $D.test$ is small, err_{test} may be less reliable as it would be computed over a small number of test instances. Moreover, err_{test} can have a high variance as we change the random partitioning of D into $D.train$ and $D.test$.

We can of course repeat the holdout method several times to obtain a distribution of the test error rates. This approach is known as *random subsampling* and produces a distribution of the error rates that can be used to understand the variance of err_{test}.

- Cross-validation

This widely-used model evaluation method aims to make effective use of all labelled instances in D for both training and testing. To demonstrate, suppose we are given a labelled set that we have randomly partitioned into three equal-sized subsets, S_1, S_2, S_3 (see Figure 3.11). For the first run, we train a model using subsets S_2, S_3 (shown as empty blocks) and test the model on subset S_1. The test error rate on S_1, denoted by $err(S_1)$, is then computed in the first run.

Likewise, for the second run we use S_1, S_3 as the training set and S_2 as the test set, computing the test error rate $err(S_2)$. Finally, we use S_1, S_2 as the training sets, and S_3 as the test set for which we compute $err(S_3)$. The overall test error rate is obtained by

summing up the number of errors committed in each test subset across all runs and divided by the total number of instances. This approach is called *three-fold validation*.

More generally, k-fold cross-validation partitions our labelled data set D (of size N) into k equal-sized[3] partitions (or 'folds') $S_1, \ldots, S_k$. During the i^{th} run, partition S_i is reserves as the test set and the remaining partitions are collectively used to train the model. A model $m(i)$ is learned and applied on $D.test(i)$ to obtain the sum of test errors $err_{sum}(i)$. This procedure is then repeated k times and the total test error rate is given by,

$$err_{test} = \frac{\sum_i^k err_{sum}(i)}{N}.$$

Unlike the holdout and random subsampling methods, here each sample is used the same number of times $(k-1)$ for training and once for testing. Furthermore, every run uses $\frac{k-1}{k}$ fraction of the data for training and $\frac{1}{k}$ fraction for testing.

The question now becomes that of choosing k. The 'right choice' depends on a number of characteristics of the problem. A small value of k will result in a smaller training set at every run, which will result in a larger estimate of generalisation error rate than what is expected of a model trained over the entire labelled set. At the other end, a higher value of k results in a larger training set at every run which in turn reduces the bias in the estimate of generalisation error rate. In the extreme case when $k = N$, every run uses exactly one data instance for test and the remainder of the data for testing. This special case is called the *leave-one-out* approach. This approach has the advantage of utilising as much data as possible for training but is computationally expensive for large data sets (cross-validation needs to be repeated N times). Moreover, it can produce misleading results in certain scenarios, as seen in the Exercise sheet. For most practical applications, the choice of k is between 5 and 10. This provides a reasonable approach for estimating the generalisation error rate because each fold is able to make use of 80% to 90% of the labelled data for training.

The k-fold cross-validation method, as described above, produces a single estimate of the generalisation error rate. However, it does not provide any information about the variance of the estimate. To obtain this, we can run k-fold cross-validation for every possible partitioning of the data into k partitions. This produces a distribution of test error rates computed for each partitioning. The average test error rate across all possible partitioning serves as a more robust estimate of generalisation error rate and can also provide information on its variance. We call this approach *complete cross-validation*. While more robust, this approach is usually too expensive to consider. A more practical solution is to repeat the cross validation approach multiple times, using a different random partitioning of the data each time. Again, we can then use the average test error rate as the estimate of generalisation error rate.

3.6 Ensemble Methods

A popular method of improving classification is to use multiple classifiers and them aggregate the predictions. Such a technique is often known as an *ensemble* or *classifier combination* method. The basic idea is to take a set of base classifiers, perform classification and then

[3]If the set does not partition exactly, then each subset is approximately of equal size.

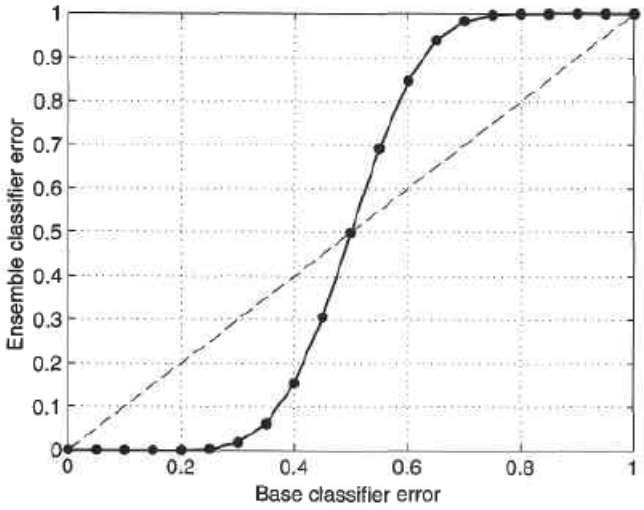


Figure 3.12: Comparison between errors of base classifiers and errors of the ensemble classifier. [2]

take a vote on the predictions made by each base classifier. The final prediction is then chosen by choosing whichever prediction wins the vote.

Let's consider an example to better understand why this will improve a classifier's performance. Suppose we have 25 binary classifiers, each of which has an error rate of $\epsilon = 0.35$. The ensemble classifier predicts the class label of a test example by taking a majority vote on the prediction made by the base classifiers. We have two options:

- The base classifiers are identical.
- The base classifiers are independent, i.e. their errors are uncorrelated.

In the former case, each base classifier will commit the same mistake and thus the error rate of the ensemble will remain at 0.35. However, for the latter case, the ensemble will predict incorrectly only if more than half the base classifiers predict incorrectly. Indeed, in this case

$$e_{ensemble} = \sum_{i=13}^{25} \epsilon^i (1-\epsilon)^{25-i} = 0.06.$$

As we can see, this is much lower than the error rate of the base classifiers.

In Figure 3.12, we see the ensemble error rate ($\epsilon_{ensemble}$) for different base classifier rates (ϵ). Again, this is for an ensemble of 25 binary classifiers. The diagonal line represents the case in which the base classifiers are identical, while the solid line represents the case in which base classifiers are independent. The key takeaway here is that the ensemble classifier performs worse than the base classifiers only when ϵ is larger than 0.5, i.e. when the base classifiers are performing worse than random guessing.

Summarising, this gives us two necessary conditions for an ensemble classifier to perform better than a single classifier:

1. The base classifiers should be independent of each other; and
2. The base classifiers should perform better than random guessing.

In practice, it is difficult to ensure total independence among the base classifiers but this is the ideal. Regardless, even when the base classifiers are somewhat correlated, ensemble methods have been shown to improve classification accuracies.

There are several methods for constructing ensemble classifiers based upon manipulating different aspects of the base classifiers or dataset under investigation. Some general approaches to this are as follows:

1. Manipulate the training set

   In this approach, we take the original data and sample it according to some sampling distribution. By repeating this, we obtain multiple training sets and can construct a classifier for each training set. In this way, we end up with our ensemble.

   Clearly, there are many ways to sample the original data and how we do this will greatly affect the ensemble since this process determines the likelihood of a given data point being chosen for testing. Two well-known ensembles that manipulate the training set are *bagging* and *boosting*.

2. Manipulate the input features

   In this approach, a subset of input features is chosen to form each training set. This subset can be chosen either randomly or based upon domain expertise. Some studies have shown this works well with datasets that contain highly redundant features.

   A classic example of this approach is the *random forest* model which uses decision trees as its base classifiers.

3. Manipulate the class labels

   In this approach, the training data is transformed into a binary class problem by randomly partitioning the class labels into two disjoint subsets A_0, A_1. Training examples whose class label belongs to the subset A_i are assigned to class i and the relabelled examples are then used to train a base classifier. By repeating this process multiple times, an ensemble of base classifiers is obtained.

   When a test example is presented, each base classifier C_i is used to predict its class label. If the test example is predicted as class 0, then all the classes that belong to A_0 will receive a vote. Likewise for class 1 and A_1. At the end, the votes are tallied and the class which receives the highest vote is assigned to the test example. Obviously, this method works better when the number of classes is sufficiently large.

   An example of this approach is the *error-correcting output coding* method.

4. Manipulate the learning algorithm

   In this approach, the learning algorithm itself is manipulated in such a way that applying the algorithm several times on the same data will result in the construction of

different classifiers. Note that not all learning algorithms can be manipulated in this way.

Two examples of this include artificial neural networks and decision trees. For the former, there can be a change to its network topology, or to the initial weights of the links between neurons. For the latter, we can inject randomness into the tree-growing procedure, such as randomly choosing one of the top k attributes for splitting, as opposed to choosing the best splitting attribute at each node.

Once an ensemble of classifiers has been learned, a test example $\mathbf{x}$ is classified by combining the predictions made by the base classifiers $C_i(\mathbf{x})$:

$$C_*(\mathbf{x}) = f(C_1(\mathbf{x}), C_2(\mathbf{x}), \ldots, C_k(\mathbf{x})),$$

where f is the function that combines the ensemble responses. A simple approach is to simply take a majority vote, but alternatives exist such as taking a weighted majority vote in which the weight of a base classifier denotes its accuracy or relevance.

N.B. Ensemble methods show the most improvement when used with *unstable classifiers*; that is, with base classifiers that are sensitive to minor perturbations in the training set, because of high model complexity.

The above point relates to the trade-off between bias and variance. Unstable classifiers often have low bias in finding the optimal decision boundary, but their predictions have a high variance for minor changes in the training set or model selection.

The process or formally analysing the generalisation error of a predictive model is called the *bias-variance decomposition*. We consider the intuition of this by discussing an analogue of a regression problem. Suppose we have a cannon at position $\mathbf{x}$ which fires projectiles at a target y.[4] Let $\hat{y}$ denote the point where the projectile hits the ground, which is analogous of the prediction of the model. Now, we want our predictions to be as close to the true target as possible. However, different trajectories of projectiles are possible based on differences in the training data or in the approach used for model selection. We can therefore observe a *variance* in the predictions $\hat{y}$ over different runs of projectile. Furthermore, the target in our example is not fixed. Instead, it has some freedom to move around and this results in a *noise* component in the true target. This can be understood as the non-deterministic nature of the output variable, where the same set of attributes can have different output values. We now let $\hat{y}_{avg}$ represent the average prediction of the projectile over multiple runs, and y_{avg} denote the average target value. The *bias* of the model is then given by $\hat{y}_{avg} - \hat{y}_{avg}$.

In this way, we can have models which produce high variance but low bias by having a wider spread of projectiles but in which the average is close to our target. This is typical of overfitting. Alternatively, we can have a model which presents low variance but high bias. In this case, our projectiles are landing in roughly the same location (low spread), but far away from our target. This is typical of underfitting.

[4]The target corresponds to the desired output at a test instance, while the starting position corresponds to its observed attributes. The projectile represents the model used for predicting the target using the observed attributes.

In the context of classification, it can be shown that the generalisation error of a classification model m can be decomposed into terms involving the bias, variance and noise components of the model:

$$gen.error(m) = c_1 \times noise + bias(m) + c_2 \times variance(m),$$

where c_1, c_2 are constants that depend upon the characteristics of training and test sets.

N.B. While the noise term is intrinsic to the target class, the bias and variance terms depend upon the choice of the classification model.

The bias of the model represents how close the average prediction of the model is to the average target. Models that are able to learn complex decision boundaries (e.g. k- nearest neighbour and multi-layer ANN) generally show low bias. The variance of a model captures the stability of its predictions in response to minor perturbations in the training set or the model selection approach.

We say that a model shows better generalisation performance if it has a *lower bias* and *lower variance*. However, if the complexity of a model is high but the training size is small, we generally expect to see a lower bias but higher variance (overfitting). An overly simplistic model may show a lower variance but would suffer from a higher bias (underfitting). Hence, the trade-off between bias and variance provides a useful way for interpreting the effects of underfitting and overfitting on the generalisation performance of a model.

The bias-variance trade-off can be used to explain why ensemble learning improves the generalisation performance of unstable classifiers. If a base classifier shows low bias but high variance, it can become susceptible to overfitting, as even a small change in the training set will result in different predictions. However, by combining the responses of multiple base classifiers, we can expect to reduce the overall variance. Hence, ensemble learning methods show better performance, primarily by lowering the variance in the predictions, although they can also help reduce the bias. One of the simplest approaches for combining predictions and reducing their variance is to compute their average. This forms the basis of the bagging method.

3.6.1 Bagging

Bagging (also known as *bootstrap aggregating*) is a method of repeatedly sampling (with replacement) a data set according to a uniform probability distribution in such a way that each bootstrap sample has the same size as the original data. On average, a bootstrap sample D_i contains approximately 63% of the original training data. To see why this is, consider the probability of a sample not being selected in D_i. We take N runs through the data (our original data has N data points) and so the probability of this is $(1 - \frac{1}{N})^N$. Thus, the probability of each data point being sampled is $1 - (1 - \frac{1}{N})^N$ and if N is sufficiently large then this probability converges to $1 - \frac{1}{e} \approx 0.632$. We summarise the bagging algorithm in Table 3.1 in which δ refers to the Kronecker delta such that $\delta(-) = 1$ if its argument is true, and 0 otherwise.

Example 3.15. *Consider the following dataset:*

Algorithm:	Bagging algorithm
Step 1	Let k be the number of bootstrap samples;

Step 2	**for** $i = 1$ **do**
	Create a bootstrap sample of size N, D_i;
	Train a base classifier C_i on the bootstrap sample D_i;
	end for

Step 3	$C_*(\mathbf{x}) = argmax_y \sum_i \delta(C_i(\mathbf{x}) = y)$.

Table 3.1: Bagging Algorithm.

x	1	2	3	4	5	6	7	8	9	10
y	1	1	1	-1	-1	-1	-1	1	1	1

We have a one-dimensional attribute x and a class label y. We suppose we only use a one-level binary decision tree, with a test condition $x \leq k$, where k is a split point chosen to minimise the entropy of the leaf nodes. Such a tree is known as a decision stump.

Without bagging, the best decision stump we can produce splits the instances at either $x \leq 3.5$ or $x \leq 7.5$. Either way, the accuracy of the tree is at most 70%. Now suppose we apply the bagging procedure on the data set using 10 bootstrap samples. These are as follows, where on the right we have described the decision stump being used for each round:

Bagging round 1

x	1	2	2	3	4	4	5	6	9	9
y	1	1	1	1	-1	-1	-1	-1	1	1

$x \leq 3.5 \Rightarrow y = 1$
$x > 3.5 \Rightarrow y = -1$

Bagging round 2

x	1	2	3	4	5	8	9	10	10	10
y	1	1	1	-1	-1	1	1	1	1	1

$x \leq 6.5 \Rightarrow y = 1$
$x > 6.5 \Rightarrow y = 1$

Bagging round 3

x	1	2	3	4	4	5	7	7	8	9
y	1	1	1	-1	-1	-1	-1	-1	1	1

$x \leq 3.5 \Rightarrow y = 1$
$x > 3.5 \Rightarrow y = -1$

Bagging round 4

x	1	1	2	4	4	5	5	7	8	9
y	1	1	1	1	-1	-1	-1	-1	1	1

$x \leq 3 \Rightarrow y = 1$
$x > 3 \Rightarrow y = -1$

Bagging round 5

x	1	1	2	5	6	6	6	10	10	10
y	1	1	1	-1	-1	-1	-1	1	1	1

$x \leq 3.5 \Rightarrow y = 1$
$x > 3.5 \Rightarrow y = -1$

Bagging round 6

x	2	4	5	6	7	7	7	8	9	10
y	1	-1	-1	-1	-1	-1	-1	1	1	1

$x \leq 7.5 \Rightarrow y = -1$
$x > 7.5 \Rightarrow y = 1$

Bagging round 7

x	1	4	4	6	7	8	9	9	9	10
y	1	-1	-1	-1	-1	1	1	1	1	1

$x \leq 7.5 \Rightarrow y = -1$
$x > 7.5 \Rightarrow y = 1$

Bagging round 8

x	1	2	5	5	5	7	7	8	9	10
y	1	1	-1	-1	-1	-1	-1	1	1	1

$x \leq 7.5 \Rightarrow y = -1$
$x > 7.5 \Rightarrow y = 1$

Bagging round 9

x	1	3	4	4	6	7	7	8	10	10
y	1	1	-1	-1	-1	-1	-1	1	1	1

$x \leq 7.5 \Rightarrow y = -1$
$x > 7.5 \Rightarrow y = 1$

Bagging round 10

x	1	1	1	1	3	3	8	8	9	9
y	1	1	1	1	1	1	1	1	1	1

$x \leq 0.5 \Rightarrow y = -1$
$x > 0.5 \Rightarrow y = 1$

We then classify the entire data given above by taking a majority vote among the predictions made by each base classifier. The results of the predictions are shown in Table 3.2. Since the class labels are ± 1*, taking the majority vote is equivalent to summing up the predicted values of* y *and examining the resulting sign. This is given in the second to last row of Table 3.2. In particular, we note that the ensemble classifier perfectly classifies all 10 examples in the original data.*

Round	$x=1$	$x=2$	$x=3$	$x=4$	$x=5$	$x=6$	$x=7$	$x=8$	$x=9$	$x=10$
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
sum	2	2	2	-6	-6	-6	-6	2	2	2
sign	1	1	1	-1	-1	-1	-1	1	1	1
true class	1	1	1	-1	-1	-1	-1	1	1	1

Table 3.2: Example of combining classifiers constructed using the bagging approach. [2]

Overall, bagging improves generalisation error by reducing the variance of the base classifiers. We note that the performance of bagging depends upon the stability of the base classifier. If a base classifier is unstable, bagging helps to reduce the errors associated with random fluctuations in the training data. However, if a base classifier is stable (i.e. robust to minor perturbations in the training set), then the error of the ensemble is primarily caused by bias in the base classifier. In this situation, bagging may not be able to improve the performance of the base classifiers significantly. It may even degrade the classifier's performance because the effective size of each training set is about 37% smaller than the original data.

3.6.2 Boosting

Boosting alters the selection of training examples by assigning weights which make certain examples more or less likely to be chosen in subsequent rounds. In this way, boosting is an iterative procedure that enables the algorithm to increasingly focus on examples that are hard to classify. The weights assigned to the training examples can be used in the following two ways:

1. They can be used to inform the sampling distribution used to draw a set of bootstrap samples from the original data.

2. They can be used to learn a model that is biased toward examples with higher weight.

Initially, the examples are assigned equal weights $\frac{1}{N}$ so that they are equally likely to be chosen for training. A sample is drawn (according to the sampling distribution) which creates a new training set. We then build a classifier from this training set and it is used to classify all the examples in the original data. The weights of the training examples are then updated at the end of each boosting round. Examples that are classified incorrectly will have their weights increased, while those which are classified correctly will have their weights decreased. This forces the classifier to focus on examples that are difficult to classify in subsequent iterations. Note also that samples which were not chosen in the previous round have a better chance of being selected in the next round since their predictions in the previous round were likely to be wrong. The final ensemble is then obtained by aggregating the base classifiers obtained from each boosting round.

Over the years, several implementations of the boosting algorithm have been developed. These algorithms differ in terms of (1) how the weights of the training examples are updated at the end of each boosting round, and (2) how the predictions made by each classifier are combined. A common implementation which we will not discuss here is called AdaBoost.

3.6.3 Random Forests

In this subsection, we discuss random forests which attempt to improve the generalisation performance by constructing an ensemble of *decorrelated* decision trees.[5] This is where bagging comes in to play by using a different bootstrap sample of the training data for

[5]We do not want correlated trees since all of these trees would sort data in the same way. Consequently, there would be no advantage to this method over a single decision tree.

learning decision trees. However, there is a key distinguishing feature of random forests in that at every internal node of a tree, the best splitting criterion is chosen from among a small set of randomly selected attributes. In this way, random forests construct ensembles of decision trees by not only manipulating training instances (by using bootstrap samples similar to bagging), but also the input attributes (by using different subsets of attributes at every internal node).

Given a training set D consisting of n instances and d attributes, the basic procedure of training a random forest classifier is as follows:

1. Construct a bootstrap sample D_i of the training set by randomly sampling n instances (with replacement) from D.

2. Use D_i to learn a decision tree T_i, as follows:

   - At every internal node of T_i, randomly sample a set of p attributes.
   - Choose an attribute from this subset that shows the maximum reduction in an impurity measure for splitting.
   - Repeat this procedure until every leaf is pure, i.e. contains only instances from a single class.

Once an ensemble of decision trees has been constructed, their average prediction (majority vote) on a test instance is used as the final prediction of the random forest.

N.B. The decision trees involved in a random forest are *unpruned*. They are allowed to grow to their largest possible size until every leaf is pure. Hence, the base classifiers of a random forest represent unstable classifiers that have low bias but high variance because of their large size.

A key property of the base classifiers learned in random forests is the lack of correlation among their model parameters and test predictions. This can be attributed to the use of an independently sampled data set D_i for learning every decision tree T_i, similar to the bagging approach. However, random forests have the additional advantage of choosing a splitting criterion at every internal node using a different (and randomly selected) subset of attributes. This property significantly helps in breaking the correlation structure, if any, among the decision trees T_i.

We can explore this further. Consider a training set involving a large number of attributes where only a small subset of attributes are strong predictors of the target class (the rest are weak indicators). For such a training set, even if we consider different bootstrap samples D_i for learning T_i, we would mostly be choosing the same attributes for splitting at internal nodes, because the weak attributes would be largely overlooked when compared with the strong predictors. This can result in a considerable correlation among the trees. However, if we restrict the choice of attributes at every internal node to a random subset of attributes, then we can ensure the selection of both strong and weak predictors, thus promoting diversity among the trees. This principle is utilised by random forests for creating decorrelated decision trees.

By aggregating the predictions of an ensemble of strong and decorrelated decision trees, random forests are able to reduce the variance of the trees without negatively impacting

their low bias. This makes random forests quite robust to overfitting. Additionally, because of their ability to consider only a small subset of attributes at every internal node, random forests are computationally fast and robust even in high-dimensional settings.

Question: How do we choose the number of attributes to be selected at each node?

This hyperparameter (labelled p) of the random forest classifier is often key as a small value of p can reduce the correlation among the classifiers, but may also reduce their strength. In contrast, a large value can improve their strength but may result in correlated trees similar to bagging. There are several common suggestions for p in the literature with the following two being quite popular:

- $p = \sqrt{d}$;
- $p = \log_2(d) + 1$.

Another option is to select p by tuning it over a validation set. Yet another method (which does not require using a separate validation set) involves computing a reliable estimate of the generalisation error rate directly during training. This is known as the *out-of-bag (oob)* estimate. The oob estimate can be computed for any generic ensemble learning method that builds independent base classifiers using bootstrap samples of the training set, e.g. bagging and random forests. We leave an exploration of this to the reader. Suffice to say, random forests have been empirically found to provide significant improvements in generalisation performance that are often comparable, if not superior, to the improvements provided by the AdaBoost algorithm. Random forests are also more robust to overfitting and run much faster than the AdaBoost algorithm.

We conclude with Table 3.3 which compares the accuracy of a decision tree classifier against three ensemble methods for 24 classic datasets. We note that the base classifiers used in each ensemble consisted of 50 decision trees and the classification accuracies are obtained from ten-fold cross-validation. Observe that the ensemble classifiers generally outperform a single decision tree classifier on many of the datasets.

Data Set	Number of (Attributes, Classes, Records)	Decision Tree (%)	Bagging (%)	Boosting (%)	RF (%)
Anneal	(39, 6, 898)	92.09	94.43	95.43	95.43
Australia	(15, 2, 690)	85.51	87.10	85.22	85.80
Auto	(26, 7, 205)	81.95	85.37	85.37	84.39
Breast	(11, 2, 699)	95.14	96.42	97.28	96.14
Cleve	(14, 2, 303)	76.24	81.52	82.18	82.18
Credit	(16, 2, 690)	85.8	86.23	86.09	85.8
Diabetes	(9, 2, 768)	72.40	76.30	73.18	75.13
German	(21, 2, 1000)	70.90	73.40	73.00	74.5
Glass	(10, 7, 214)	67.29	76.17	77.57	78.04
Heart	(14, 2, 270)	80.00	81.48	80.74	83.33
Hepatitis	(20, 2, 155)	81.94	81.29	83.87	83.23
Horse	(23, 2, 368)	85.33	85.87	81.25	85.33
Ionosphere	(35, 2, 351)	89.17	92.02	93.73	93.45
Iris	(5, 3, 150)	94.67	94.67	94.00	93.33
Labor	(17, 2, 57)	78.95	84.21	89.47	84.21
Led7	(8, 10, 3200)	73.34	73.66	73.34	73.06
Lymphography	(19, 4, 148)	77.03	79.05	85.14	82.43
Pima	(9, 2, 768)	74.35	76.69	73.44	77.60
Sonar	(61, 2, 208)	78.85	78.85	84.62	85.58
Tic-tac-toe	(10, 2, 958)	83.72	93.84	98.54	95.82
Vehicle	(19, 4, 846)	71.04	74.11	78.25	74.94
Waveform	(22, 3, 5000)	76.44	83.30	83.90	84.04
Wine	(14, 3, 178)	94.38	96.07	97.75	97.75
Zoo	(17, 7, 101)	93.07	93.07	95.05	97.03

Table 3.3: Comparing the accuracy of a decision tree classifier against three ensemble methods. [2]

Chapter 4

Cluster Analysis

4.1 Preamble

Cluster analysis, or simply *clustering* is the process of partitioning a set of data objects into subsets. Each subset is a *cluster* in which all objects in the cluster are similar to one another, yet dissimilar to objects in other clusters. The process of clustering is the way in which we construct such clusters. Clustering is also known as *unsupervised learning* because class label information is not present. It is a form of *learning by observation*, rather than *learning by examples*. In contrast, in classification is known as *supervised learning* because the class label information is given, i.e. the learning algorithm is supervised in that it is told the class membership of each training example.

There are several different ways to construct clusters, and different clustering methods may generate different clusterings on the same data set. As a rule, the partitioning is not performed by humans, but instead by clustering algorithms. In this opening section we will first discuss some examples to motivate our interest in clustering, before concluding the section with an outline of different clustering strategies.

- Clustering for understanding

  Often, we want to create clusters so that we can obtain some understanding of the data under investigation. The following are some examples:

  - In biology, a well-known task is the construction of a taxonomy (hierarchical classification) of all living things: kingdom, phylum, class, order, family, genus, and species. Much of the early work in cluster analysis sought to create a discipline of mathematical taxonomy that could automatically find such classifications. More recently, biologists have applied clustering to analyse the large amounts of genetic information now available, e.g. using clustering to find groups of genes that have similar functions.
  - Information retrieval problems can be used in algorithms designed to find certain objects of interest on the internet. For example, a keyword search often returns a very large number of hits (by which we mean pages relevant to the search). This is due to the extremely large number of web pages. Clustering can be used to organise the search results into groups and present the results in a concise and

easily accessible way. For instance, a query of 'movie' might return web pages grouped into categories such as *reviews, trailers, stars, and theatres.* Each category (cluster) can then be further broken down into subcategories (subclusters), producing a hierarchical structure that further assists a user's exploration of the query results.

- Understanding the Earth's climate requires finding patterns in the atmosphere and ocean. To that end, cluster analysis can be applied to find patterns in atmospheric pressure and ocean temperature that have a significant impact on climate.
- An illness or condition frequently has a number of variations. Cluster analysis can be used to identify these different subcategories. For example, clustering can be used to identify different types of depression. Cluster analysis can also be used to detect patterns in the spatial and/or temporal distribution of a disease.
- In business intelligence, clustering can be used to organise a large number of customers into groups, where customers within a group share strong similar characteristics. This facilitates the development of business strategies for enhanced customer relationship management. Another example might be a consultant company with a large number of projects. To improve project management, clustering can be applied to partition projects into categories based on similarity so that project auditing and diagnosis (to improve project delivery and outcomes) can be conducted effectively.

- Clustering for utility

  Cluster analysis provides an abstraction from individual data objects to the clusters in which those data objects reside. Additionally, some clustering techniques characterise each cluster in terms of a cluster prototype, i.e. a representative data object in a given cluster. These prototypes can then be used as the basis for a number of additional data analysis or data processing techniques. Therefore, in the context of utility, cluster analysis is the study of techniques for finding the most representative cluster prototypes.

  - In image recognition, clustering can be used to discover clusters or subclasses in handwritten character recognition systems. Suppose we have a data set of handwritten digits, where each digit is labelled as either 1, 2, 3 and so on. There is of course a large variation in which people write the same digit (e.g. some people have a small loop in the bottom left corner when writing '2') and so using multiple models based on the subclasses can improve overall recognition accuracy.
  - Many data analysis, such as regression or PCA, have a time/space complexity of $O(m^2)$ or higher (where m is the number of objects). For large data sets, these are therefore not practical. One solution to this is to apply the algorithms to a reduced data set consisting only of cluster prototypes. Depending on the type of analysis, the number of prototypes, and the accuracy with which the prototypes represent the data, the results can be comparable to those that would have been obtained if all of the data had be used.

- Cluster prototypes can also be used for data compression. In particular, a table is created consisting of the prototypes for each cluster. Each prototype is then assigned an integer value that is its index/position in the table. This type of compression is known as *vector quantization* and is often applied to image, sound and video where
  1. many of the data objects are highly similar to one another,
  2. some loss of information is acceptable, and
  3. a substantial reduction in the data size is desired.
- Finding nearest neighbours can require computing the pairwise distance between all points and this can be computationally expensive. A more efficient approach is to instead compute pairwise distances between clusters and their cluster prototypes. Intuitively, if two cluster prototypes are far apart, then the objects in the corresponding clusters cannot be nearest neighbours of each other.
- Because a cluster is a collection of data objects that are similar to one another within the cluster and dissimilar to objects in other clusters, a cluster of data objects can be treated as an implicit class. In this sense, clustering is sometimes called *automatic classification*.
- Clustering is also sometimes called data segmentation because clustering partitions large data sets into groups according to their similarity. Clustering can therefore be used for *outlier* detection, where outliers may be more interesting than common cases.

With the above motivating us for the remainder of this chapter, we next provide an overview of basic clustering strategies. Note that, in general, it is difficult to provide a crisp categorisation of clustering methods because categories inevitably overlap.

- Partitioning methods

  Suppose we are given a set of n objects. A partitioning method constructs k partitions of the data, where each partition represents a cluster and $k \leq n$ (i..e each partition must contain at least one object). This last requirement can be relaxed in certain scenarios, such as those of *fuzzy partitioning techniques*.

  Most partitioning methods are distance based. Given k (the number of partitions to construct), a partitioning method creates an initial partitioning. It then uses an *iterative relocation technique* that attempts to improve the partitioning by moving objects from one group to another. The general criterion of a good partitioning is that objects in the same cluster are 'close', while objects in different clusters are 'far apart'. This is not well-defined and the specifics will change depending on the situation. In any case, traditional partitioning methods can be extended for subspace clustering, rather than searching the full data space. This is useful when there are many attributes and the data are sparse.

  Achieving global optimality in partitioning-based clustering is often computationally prohibitive, potentially requiring an exhaustive enumeration of all the possible partitions. Instead, most applications adopt popular heuristic methods, such as greedy

approaches like the k-means and the k-medioids algorithms. These progressively improve the clustering quality and approach a local optimum. These heuristic clustering methods work well for finding spherical-shaped clusters in small-to-medium-size datababases.

- Hierarchical methods

  A hierarchical method creates a hierarchical decomposition of the given data set. Such a method can be classified as being either *agglomerative* or *divisive*, based on how the hierarchical decomposition is formed.

  - The *agglomerative approach* is also called a *bottom-up approach*. It starts with each object forming a separate group and successively merges the objects or groups. This continues until there is just one topmost level of the hierarchy, or a termination condition is applied.
  - The *divisive approach* is also called a *top-down* approach. It starts with each object in the same cluster and then successively splits the cluster(s) into smaller clusters until eventually each object is in just one cluster, or a termination condition is applied.

  Hierarchical clustering methods can be distance-based, density-based or continuity-based. Various extensions of hierarchical strategies consider clustering in subspaces as well.

  A potential drawback of hierarchical strategies is that once a merge/split is performed, it can never be undone. Nevertheless, this rigidity is useful in the sense that is leads to smaller computation costs by not having to worry about a combinatorial number of different choices.

- Density-based methods

  Most partitioning methods cluster objects based on the distance between objects. Such methods can find only spherical-shaped clusters and encounter difficulty in discovering clusters of arbitrary shapes. Other strategies have been developed based on the idea of density. The general idea is to continue growing a given cluster as long as the density (number of data points) in the neighbourhood exceeds some threshold. For example, for each data point within a given cluster, the neighbourhood of a given radius has to contain at least a minimum number of points. Such methods can be used to filter out noise or outliers and discover clusters of arbitrary shape.

- Grid-based methods

  Grid-based methods quantise the object into a finite number of cells that form a grid structure. All the clustering operations are performed on the grid structure. All the clustering operations are performed on the grid structure (i.e. on the quantised space). The main advantage of this approach is its fast processing time, which is typically independent of the number of data objects and dependent only on the number of cells in each dimension in the quantised space.

Using grids is often an efficient approach to many spatial data mining problems, including clustering. therefore, grid-based methods can be integrated with other clustering methods such as density-based methods and hierarchical methods.

4.2 K-means Clustering

The simplest and most fundamental type of cluster analysis is partitioning. This organises the objects into several exclusive clusters, the number of which is usually assumed to be given. So, in general, we have a data set D with n objects for which we wish to partition into k (where $k \leq n$) clusters. The partitioning algorithm then organises these n objects into the k partitions and, in principle, does so in a way which optimises the objective partitioning criterion, which could be a dissimilarity function based on distance, say. In this way, we have k clusters $C_1, \ldots, C_k$, each of which are contained in D and for which $C_i \cap C_j = \emptyset$ when $i \neq j$. If our objective function is well-chosen, we have high intracluster similarity and low intercluster similarity.

There are several algorithms we can use, but two of the most prominent are K-means K-medoid. Both are prototype-based clustering algorithms with the former defining a prototype in terms of a centroid. This is usually the mean of a group of points and is typically applied to objects in a continuous n-dimensional space. In contrast, K-medoid clustering defined a prototype in terms of a medoid, which is the most representative point for a group of points. This can be applied to a wide range of data since it requires only a proximity measure for a pair of points. While a centroid almost never corresponds to an actual data point, a medoid, by its definition, must always be an actual data point.

Our focus will now be on the K-means algorithm, whose basic algorithm we now describe:

1. First, we choose k initial centroids, where k is our desired number of clusters.

2. Each point is then assigned to the closest centroid, and each collection of points assigned to a centroid is a cluster.

3. The centroid of each cluster is then updated based on the points assigned to the cluster.

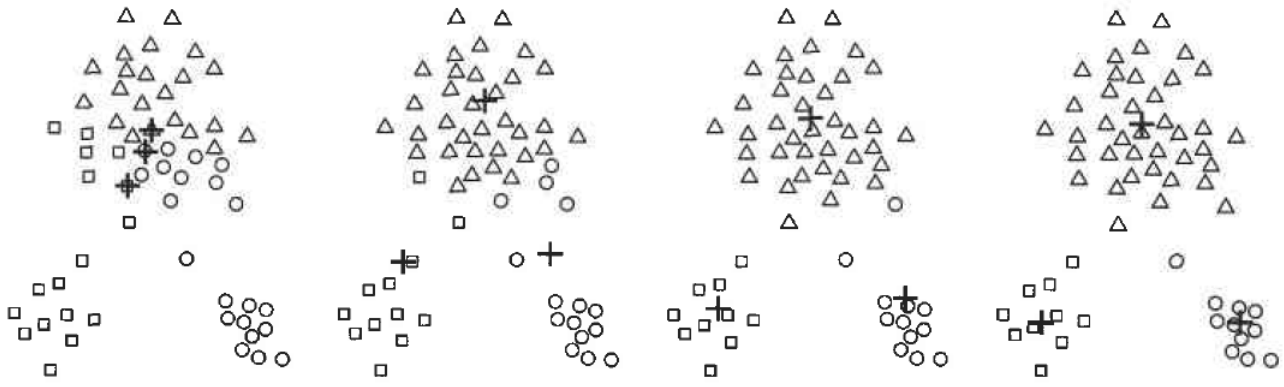
4. We repeat the assignment and update steps until no point changes clusters, or equivalently, until the centroids remain the same.

Formally, this is described in Table 4.1 and is visualised in Figure 4.1. Here, we start with three centroids and proceed in four steps. Note that the centroids are indicated by '+', and all points belonging to the same cluster have the same marker shape. In the first step, points are assigned to the initial centroids, which are all in the larger group of points. After the points are assigned to the centroid, the centroid is then updated. In the second step, the points are assigned to the updated centroids before the centroids are updated again. We note that in Steps 2, 3 and 4, two of the centroids move to the two small groups of points at the bottom of the figures. When the K-means algorithm terminates in Step 4 (because no changes occur), the centroids have identified the natural groupings of points.

For a number of combinations of proximity functions and types of centroids, K-means always converges to a solution (i.e. K-means reaches a state in which no points are shifting

Input:	k: the number of clusters, D: a data set containing n objects
Output:	A set of k clusters
Method:	
Step 1	Arbitrarily choose k objects from D as the initial cluster centres;
Step 2	**repeat**
Step 3	(re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
Step 4	update the cluster means; that is, calculate the mean value of the objects for each cluster;
Step 5	**until** no change;

Table 4.1: Basic K-means Algorithm.



(a) Iteration 1. (b) Iteration 2. (c) Iteration 3. (d) Iteration 4.

Figure 4.1: Using the K-means algorithm to find three clusters in sample data. [2]

from one cluster to another, and hence the centroids don't change). However, because most of the convergence occurs in the early steps, we often replace the condition in Step 5 of Table 4.1 by a weaker condition, e.g. repeat until only 1% of the points change clusters.

Example 4.1. *Consider a set of objects located in two-dimensional space (something like Figure 4.1) and let $k = 3$, i.e. we want to partition our data set into three clusters. According to our algorithm, we arbitrarily choose three objects as the three initial cluster centres, where cluster centres are marked by a +. Each object is then assigned to a cluster based on the cluster centre to which it is nearest.*

Next, the cluster centres are updated. For instance, the mean value of each cluster is recalculated based on the current objects in the cluster. Using these new cluster centres, the objects are redistributed to the clusters based on which cluster centre is nearest. This process iterates, continuing to reassign objects to clusters to improve the partitioning. This is often referred to as iterative relocation. Eventually, no reassignment of the objects in any cluster occurs (or a sufficiently small number of points are reassigned) and so the process terminates. The resulting clusters are returned by the clustering process.

With the above in mind, we now look at each step of the basic K-means algorithm in more detail. First, we look at **Step 2** and assigning points to the closest centroid. To do this, we need a proximity measure that quantifies the notion of 'closest' for the specific data under consideration. The typical choice when working in Euclidean space is the Euclidean L_2 distance, while for documents we often use the cosine similarity. Of course, we can also use any of the other proximity measures discussed in Section 2.5, such as the Manhattan L_1 distance or Jaccard similarity measure. The former would be used for Euclidean data, while the latter for documents.

As the algorithm repeatedly calculates similarity, it is typical to use relatively simple similarity measures. However, in some cases, such as in low-dimensional Euclidean space, it is possible to avoid computing many of the similarities, thereby significantly speeding up the K-means algorithm. Another approach to speed up the algorithm is the bisecting K-means strategy which we will discuss towards the end of this chapter. This speeds things up by reducing the number of similarities computed.

Skipping ahead to **Step 4**, we should note that the centroid can vary depending on the proximity measure for the data and the goal of the clustering. The goal of the clustering is typically expressed by an objective function which depends on the proximities of the points to one another, or to the cluster centroids. For instance, we may wish to minimise the squared distance of each point to its closest centroid. We will now demonstrate this with two examples, but in both the key point is that once we have specified a proximity measure and an objective function, the centroid that we should choose can be determined mathematically.

Example 4.2 (Data in Euclidean Space)**.** *Consider data whose proximity measure is Euclidean distance. For our objective function, which measures the quality of a clustering, we use the sum of the squared error (SSE). Here, we calculate the error of each data point (its Euclidean distance to the closest centroid) and then compute the total sum of the squared errors. Given two different sets of clusters that are produced by two different runs of K-means, we choose the one with smallest squared error. This results in the prototypes (centroids)*

of this clustering being a better representation of the points in their cluster. In this way, clustering becomes an optimisation problem.

We now make the following notational choices:

- $\underset{\sim}{x}$*, an object*
- C_i*, the* i^{th} *cluster,*
- $\underset{\sim}{c_i}$*, the centroid of cluster* C_i*,*
- $\underset{\sim}{c}$*, the centroid of all points,*
- m_i*, the number of objects in the* i^{th} *cluster,*
- m*, the number of objects in the data set,*
- k*, the number of clusters.*

Given this notation, the SSE is formally defined as follows:

$$SSE = \sum_{i=1}^{k} \sum_{\underset{\sim}{x} \in C_i} dist(\underset{\sim}{c_i}, \underset{\sim}{x})^2,$$

where dist is the standard Euclidean L_2 *distance between two objects in Euclidean space. We now want to minimise the total SSE, however doing so is typically computationally infeasible. As such, more practical approaches, such as the gradient descent method, are used.*

To keep things simple, we consider the case when the proximity function is Euclidean distance and the objective is to minimise the SSE in the case of one-dimensional data. In other words, we want to minimise,

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} (c_i - x)^2,$$

where C_i *is the* i^{th} *cluster,* x *is a point in* C_i *and* c_i *is the mean of the* i^{th} *cluster.*

We solve for the j^{th} *centroid* c_j*, which minimises this equation, by differentiating the SSE and equating to 0. First,*

$$\begin{aligned} \frac{\partial}{\partial c_j} SSE = & \ \tfrac{\partial}{\partial c_j} \textstyle\sum_{i=1}^{k} \sum_{x \in C_i} (c_i - x)^2 \\ = & \ \textstyle\sum_{i=1}^{k} \sum_{x \in C_i} \tfrac{\partial}{\partial c_j} (c_i - x)^2 \\ = & \ \textstyle\sum_{x \in C_j} \tfrac{\partial}{\partial c_j} (c_j - x)^2 \\ = & \ \textstyle\sum_{x \in C_j} 2(c_j - x) \qquad = 0. \end{aligned}$$

We want to solve this for c_j *and recall that we said* m_j *is the number of objects in the* i^{th} *cluster (i.e. we are summing* m_j *times). Rearranging therefore gives,*

$$m_j c_j = \sum_{x \in C_k} x \ \Rightarrow \ c_j = \frac{1}{m_k} \sum_{x \in C_j} x.$$

In other words, the best centroid for minimising the SSE of a cluster is the mean of the points in the cluster.

More generally, using our notation above, the centroid of the i^{th} cluster is given by,

$$\underset{\sim}{c}_i = \frac{1}{m_i} \sum_{\underset{\sim}{x} \in C_i} \underset{\sim}{x}.$$

To illustrate this, suppose our cluster has three points:

$$\begin{pmatrix} 1 \\ -1 \end{pmatrix}, \begin{pmatrix} 2 \\ -3 \end{pmatrix}, \begin{pmatrix} 3 \\ -5 \end{pmatrix}.$$

Then,

$$\underset{\sim}{c}_i = \frac{1}{3}\left(\begin{pmatrix} 1 \\ -1 \end{pmatrix} + \begin{pmatrix} 2 \\ -3 \end{pmatrix} + \begin{pmatrix} 3 \\ -5 \end{pmatrix}\right) = \frac{1}{3}\begin{pmatrix} 1+2+3 \\ (-1)+(-3)+(-5) \end{pmatrix} = \begin{pmatrix} 2 \\ -3 \end{pmatrix}$$

Steps 3 and 4 of the K-means algorithm directly attempt to minimise the objective function (in our case, the SSE). Step 3 then forms clusters by assigning points to their nearest centroid, which minimises the SSE for the given set of centroids. Finally, Step 4 recomputes the centroids so as to further minimise the SSE. However, the actions of K-means in Steps 3 and 4 are guaranteed to only find a local minimum with respect to the SSE because they are based on optimising the SSE for specific choices of the centroids and clusters, rather than for all possible choices.

Example 4.3 (Document Data)**.** *The K-means algorithm is in no means restricted only to data in Euclidean space. For example, if we have document data then we can use the cosine similarity measure. We assume that the document data is represented as a document-term matrix. Our objective is to maximise the similarity of the documents in a cluster to the cluster centroid. This quantity is known as the cohesion of the cluster and, as with the Euclidean data, the cluster centroid is the mean.*

The analogous quantity to the total SSE is the total cohesion, given by,

$$Total\ Cohesion = \sum_{i=1}^{k} \sum_{\underset{\sim}{x} \in C_i} cosine(\underset{\sim}{x}, \underset{\sim}{c}_i).$$

In general, there are several choices for the proximity function, centroid and objective function that can be used in the basic K-means algorithm and that are guaranteed to converge. Some possible choices (including the two just discussed) are as follows:

Common choices for proximity, centroids and objective functions		
Proximity Function	Centroid	Objective Function
Manhattan (L_1)	Median	Minimise sum of the $L-1$ distance of an object to its cluster centroid
Squared Euclidean (L_2^2)	Mean	Minimise sum of the squared L_2 distance of an object to its cluster centroid
Cosine Similarity	Mean	Maximise sum of the cosine similarity of an object to its cluster centroid
Bregman Divergence	Mean	Minimise sum of the Bregman divergence of an object to its cluster centroid

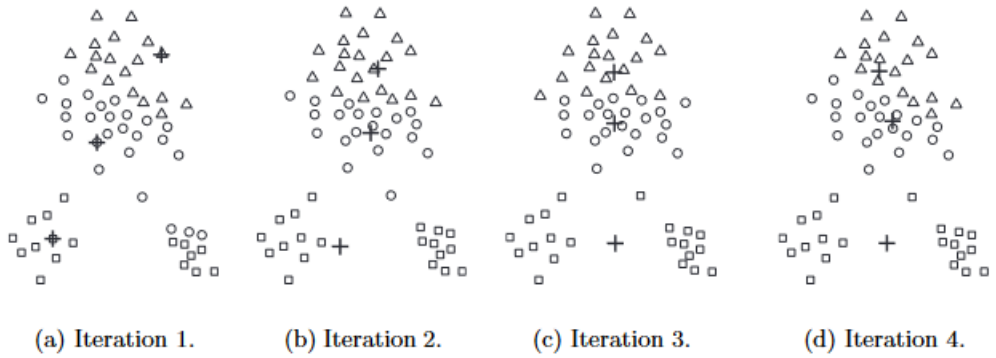


Figure 4.2: Poor starting centroids for K-means. [2]

Moving on, while the clustering algorithm can be used for a wide variety of data types of varying dimensions, we will focus simply on two-dimensional data. This is for simplicity and does not suggest any greater importance of this case.

Jumping now to Step 1, when random initialisation of centroids is used, different runs of K-means typically produce different total SSEs. As such, the K-means algorithm is not guaranteed to converge to the global optimum, and often terminates at a local optimum. Such results inevitably depend on the initial random selection and so it follows that choosing the proper initial centroids is the key step of the basic K-means procedure. While common approach is to choose the initial centroids randomly, the resulting clusters are often poor.

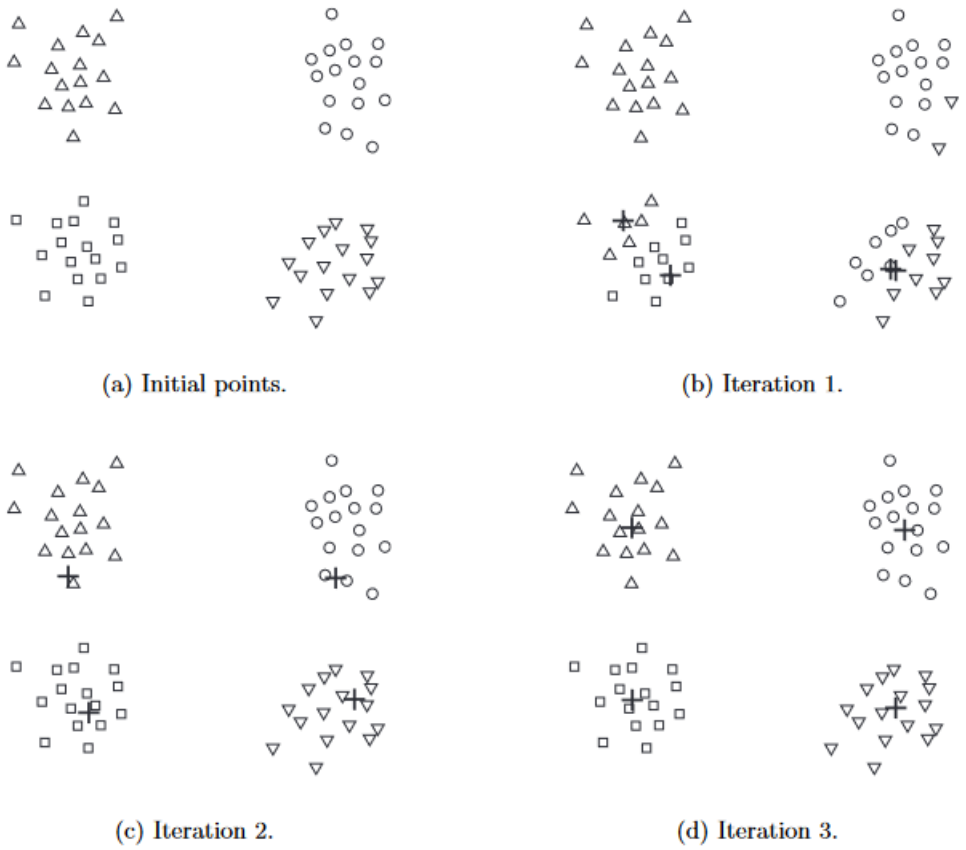
Example 4.4 (Poor Initial Centroids)**.** *Using the same data set seen in Figure 4.1, we can demonstrate how randomly selected initial centroids can be poor. Contrast Figure 4.1 with Figure 4.2. Both show clusters that result from two particular choices of initial centroids. In Figure 4.1, even though all the initial centroids are from one natural cluster, the minimum SSE clustering is still found. In Figure 4.2, however, even though the initial centroids appear better distributed, we obtain a suboptimal clustering with higher squared error.*

Example 4.5 (Multiple run-throughs)**.** *One approach to overcoming the previous example's problems is to perform multiple runs of the algorithm, each time with a different set of randomly chosen initial centroids. Then, we select the set of clusters with the minimum SSE. While simple, this strategy can struggle depending on the data set and/or the number of clusters sought.*

Consider the data set shown in Figure 4.3(a), which consists of two pairs of clusters. The clusters in each (top-bottom) pair are closer to each other than to the clusters in the other pair. Figures 4.3(b)-(d) shows that if we start with two initial centroids per pair of clusters, then even when both centroids are in a single cluster, the centroids will redistribute themselves so that the 'true' clusters are found.

However, if we next look at Figure 4.4, then we see that if a pair of clusters has only one initial centroid and the other pair has three, then two of the true clusters will be combined and one true cluster will be split.

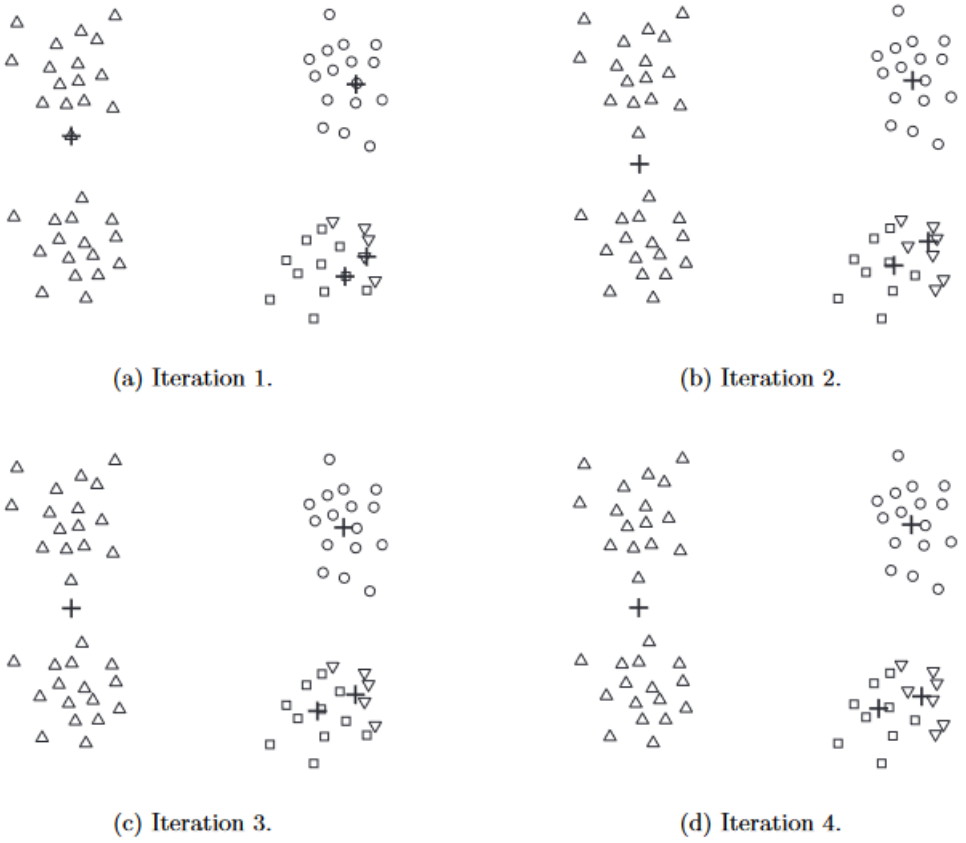
Note that an optimal clustering will only be obtained as long as two initial centroids fall anywhere in a pair of clusters, since the centroids will redistribute themselves, one to each



(a) Initial points. (b) Iteration 1.

(c) Iteration 2. (d) Iteration 3.

Figure 4.3: Two pairs of clusters with a pair of initial centroids within each pair of clusters. [2]



(a) Iteration 1. (b) Iteration 2.

(c) Iteration 3. (d) Iteration 4.

Figure 4.4: Two pairs of clusters with more or fewer than two initial centroids within a pair of clusters.

cluster. Unfortunately, as the number of clusters becomes larger, it is increasingly likely that at least one pair of clusters will have only one initial centroid.

Due to the problems highlighted above, other techniques are often employed for initialisation. We now present two approaches that can be effective.

1. Take a sample of points and cluster them using a hierarchical clustering technique. Next, k clusters are extracted from the hierarchical clustering and the centroids of those clusters are used as the initial centroids. This approach often works well but is practical only if

   (a) the sample is relatively small, e.g. a few hundred/thousand (hierarchical clustering is expensive), and

   (b) k is relatively small compared to the sample size.

2. Select the first point at random or take the centroid of all points. Then, for each successive initial centroid, select the point that is farthest from any of the initial centroids already selected. In this way, we obtain a set of initial centroids that is guaranteed to be not only randomly selected, but also well separated.

   Alas, such an approach can select outliers, rather than points in dense regions (clusters). This can lead to a situation where many clusters have just one point (an outlier) which reduces the number of centroids for forming clusters for the majority of points. As might be expected, this is also expensive.

   To overcome these problems, this approach is often applied to a sample of the points. Because outliers are comparatively rare, they tend to not show up in a random sample. In contrast, points from every dense region are likely to be included unless the sample size is very small. Furthermore, the computation involved in finding the initial centroids is greatly reduced because the sample size is typically much smaller than the number of points.

More recently, a new approach for initialising K-means has been developed, the so-called K-means++ method. This procedure is guaranteed to find a K-means clustering solution that is optimal to within a factor of $O(\log(k))$. In practice, this means noticeably better clustering results in terms of lower SSE.

The idea is similar to that just discussed; namely, we still want to pick the first centroid at random and then pick each remaining centroid as the point as far from the remaining centroids as possible. For K-means ++, the algorithm picks centroids incrementally until k centroids have been picked. At every such step, each point has a probability of being picked as the new centroid that is proportional to the square of its distance to its closest centroid. Because outliers are rare, this means it is unlikely to choose outliers for centroids. The initialisation step of the algorithm is given in Table 4.2. The rest is the same as Table 4.1.

Shortly, we will see two other approaches that are useful for producing better-quality (lower SSE) clusterings. One will be a variant of K-means called *bisecting K-means*, which is less susceptible to initialisation problems. The other will be to use postprocessing to 'fix

Method:	
Step 1	For the first centroid, pick one of the points at random;
Step 2	**for** $i = 1$ to *number of trials* **do**
Step 2a	Compute the distance $dist(x)$ of each point to its closest centroid;
Step 2b	Assign each point a probability proportional to each point's $dist(x)^2$;
Step 2c	Pick new centroid from the remaining points using the weighted probabilities.
Step 3	**end for**

Table 4.2: K-means++ initialisation Algorithm.

up' the set of clusters produced. Of course, either could be combined with the K-means++ approach.

As a final comment on the K-means algorithm, we briefly discuss the time and space requirements. In both cases these are modest since only the data points and centroids are stored.

- The storage required is $O((m + k)n)$, where
  - m is the number of points,
  - n is the number of attributes,
  - k is the number of clusters.
- The time required is $O(ikmn)$, where
  - m, n, k are as above,
  - i is the number of iterations.

Normally, $k << m$ and $i << m$ and the method is relatively scalable and efficient in processing large data sets.

4.3 Additional Issues with K-means

- Handling empty clusters

  One potential problem occurs if no points are allocated to a cluster during the assignment step. This is possible and if/when it does, a strategy is needed to choose a replacement centroid. If not, the squared error will be larger than necessary.

  - One approach is to choose the point that is farthest away from any current centroid. At the very least, this eliminates the point that currently contributes most to the total squared error.
  - Another approach is to use a K-means++ approach.
  - A third approach is to choose the replacement centroid at random from the cluster that has the highest SSE.

- Outliers

  When the squared error criterion is used, outliers can unduly influence the clusters that are found. In particular, when outliers are present, the resulting cluster centroids are typically not as representative as they otherwise would be, thereby leading to higher SSE. Because of this, it is often useful to discover outliers and eliminate them beforehand.

  Of course, there are times when we do not wish to eliminate outliers (e.g. when using clustering for data compression, financial analysis etc.). In such cases, the issue becomes one of identifying outliers. There are several techniques used for this, but these will not be the topic of these notes.

- Reducing the SSE with postprocessing

  An obvious way to reduce the SSE is to find more clusters. However, in many cases we would like to improve the SSE *without* increasing k. As K-means typically converges to a local minimum, this is often possible. There are various techniques used and the usual strategy is to focus on individual clusters since the total SSE is simply the sum of the SSE contributed by each cluster (for this reason, the terms *total* SSE and *cluster* SSE are often used).

  We can change the total SSE by performing various operations on the clusters, such as splitting or merging clusters. A commonly used approach is to employ alternate cluster splitting and merging phases. During a splitting phase, clusters are divided, while during a merging phase the are combined. In this way, it is often possible to escape local SSE minima and still produce a clustering solution with the desired number of clusters. The following are some techniques used in the splitting and merging phases.

  - Decrease the total SSE by increasing the number of clusters
    - Split a cluster
      The cluster with the largest SSE is usually chosen, but we could also split the cluster with the largest standard deviation for a particular attribute.
    - Introduce a new cluster centroid
      Often, the point that is farthest from any cluster centre is chosen. We can easily determine this if we keep track of the SSE contributed by each point. Alternatively, we could choose randomly from all points or from the points with the highest SSE with respect to their closest centroids.
  - Decrease the number of clusters while trying to minimise the increase in total SSE.
    - Disperse a cluster
      This is accomplished by removing the centroid that corresponds to the cluster and reassigning the points to other clusters. Ideally, the cluster being dispersed should be the one that increases the total SSE the least.
    - Merge two clusters

Input:	k: the number of clusters, D: a data set containing n objects
Output:	A set of k clusters
Method:	
Step 1	Initialize the list of clusters to accommodate the cluster consisting of all points;
Step 2	**repeat**
Step 2a	Remove a cluster from the list of clusters,
Step 2b	{Perform several 'trial' bisections of the chosen cluster};
Step 2c	**for** $i = 1$ to *number of trials* **do**
Step 2d	Bisect the selected cluster using basic K-means;
Step 2e	**end for**;
Step 2f	Select the two clusters from the bisection with the lowest total SSE;
Step 2g	Add these two clusters to the list of clusters;
Step 3	**until** The list of clusters contains k clusters.

Table 4.3: Bisecting K-means Algorithm.

The clusters with the closest centroids are typically chosen, although an alternative (perhaps better) approach is to merge the two clusters that result in the smallest increase in total SSE.

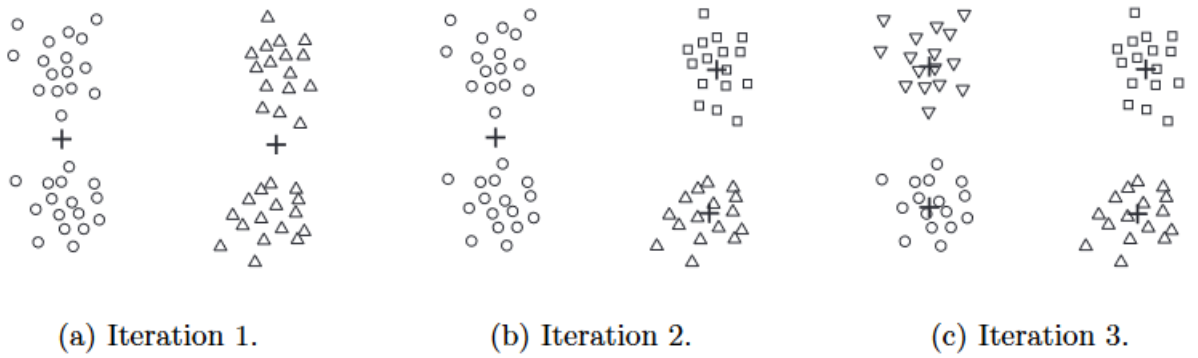
4.4 Bisecting K-means

A useful extension of the basic K-means algorithm is the *bisecting K-means algorithm*. This is a hybrid of partitional and hierarchical clustering and can recognise clusters of any size or shape. It also works well when k is large and, in general, performs better on entropy measurement. The idea is to split the set of all points into two clusters, select one of these clusters to split, and so on, until K clusters have been produced. The details of this is given in Table 4.3.

There are several ways we can choose which cluster to split. We could choose the largest cluster at each step, the cluster with the highest SSE, or use some other criterion. Suppose we use SSE and $k = 3$. Then first, all data points are put into a single cluster, C. Next, C is split into two clusters: C_1 and C_2. We haven't got three clusters yet, so we need to split again. We look at which has the higher SSE. Suppose C_1 has the higher SSE. In this case, we split C_1 into C_1' and C_1''. We now have three clusters. Of course, if we chose a different choice than SSE, we would end up with different clusters.

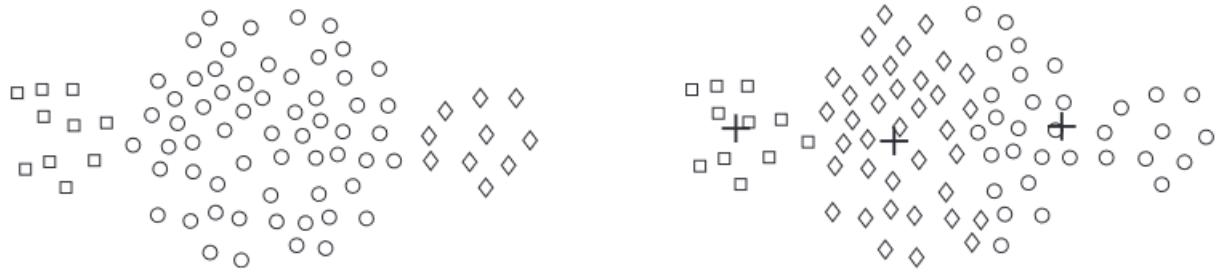
Note that since we are using the K-means algorithm 'locally' (i.e. to bisect individual clusters), the final set of clusters does not represent a clustering that is a local minimum with respect to the total SSE. Thus, we often refine the resulting clusters by using their cluster centroids as the initial centroids for the standard K-means algorithm.

Example 4.6. *Bisecting K-means is less susceptible to initialisation problems. To show this, we look at Figure 4.5. In this case, $k = 4$ and in Iteration 1, two pairs of clusters*



(a) Iteration 1. (b) Iteration 2. (c) Iteration 3.

Figure 4.5: Bisecting K-means on four clusters. [2]



(a) Original points. (b) Three K-means clusters.

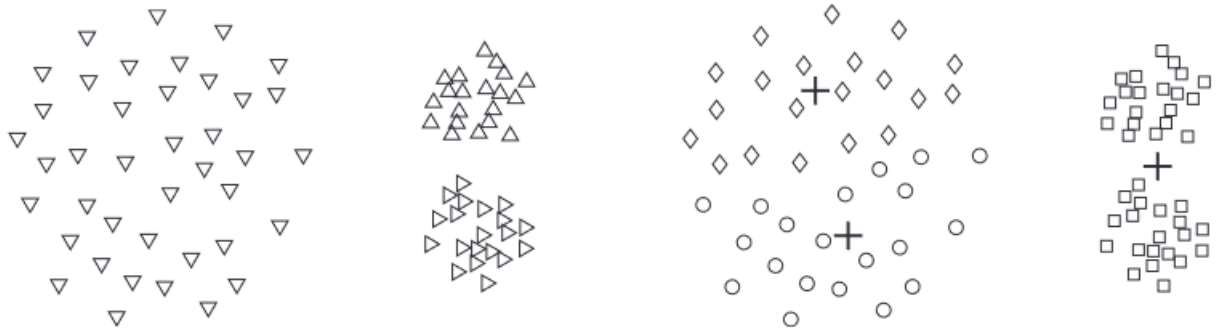
Figure 4.6: K-means with clusters of different sizes. [2]

are found. In Iteration 2, we next split the right-hand pair of clusters, before the left-hand clusters are split in Iteration 3. This results in 4 clusters, as required.

As bisecting K-means performs several trial bisections, and takes the one with the lowest SSE, this algorithm has less trouble with initialisation. The fact that there are only two centroids at each step also helps with this.

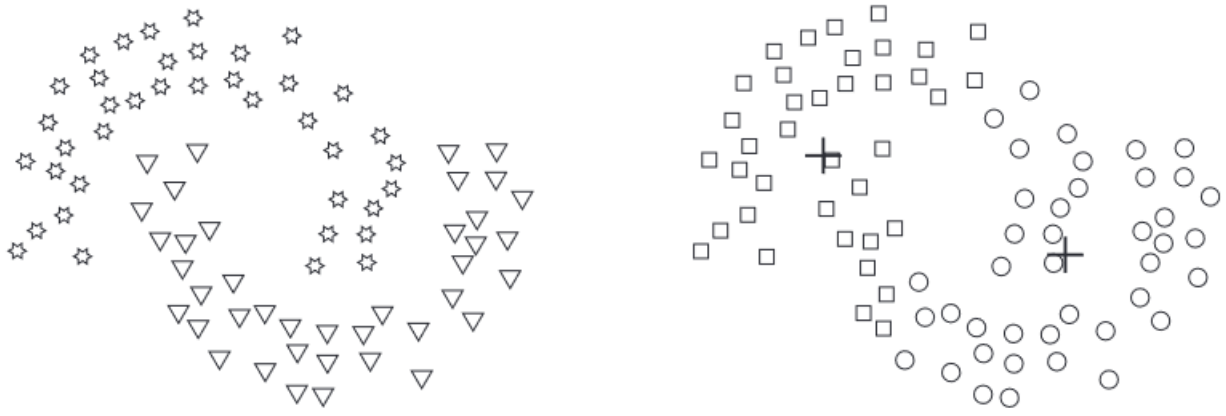
4.5 Some Concluding Remarks

Note that K-means has trouble detecting 'natural' clusters when these are non-spherical, or when clusters have widely different sizes or densities. This is visualised in Figures 4.6, 4.7, 4.8. In Figure 4.6, K-means cannot find the three natural clusters because one of the clusters (the central one) is much larger than the other two. Thus, the larger cluster is broken, while one of the smaller clusters is combined with a portion of the larger cluster. In Figure 4.7, K-means fails because the two smaller clusters are much denser than the larger cluster. Finally, in Figure 4.8, K-means finds two clusters that mix portions of the two natural clusters because it is trying to find spherical clusters, but the natural clusters have a spiral shape.



(a) Original points. (b) Three K-means clusters.

Figure 4.7: K-means with clusters of different densities. [2]



(a) Original points. (b) Two K-means clusters.

Figure 4.8: K-means with clusters that are non-spherical. [2]

The difficulties presented here arise because the K-means objective function is a mismatch for the kinds of clusters we are trying to find. This is because it is minimised by globular clusters of equal size and density, or by clusters that are well-separated. To overcome these limitations, we must be willing to accept a clustering that breaks the natural clusters into a number of subclusters. This is shown in Figure 4.9, which demonstrates what happens to our three data sets if instead we find six clusters instead of two or three. Each smaller cluster is pure in the sense that it contains only points from one of the natural clusters.

We conclude by discussing some strengths and weaknesses of the K-means algorithm.

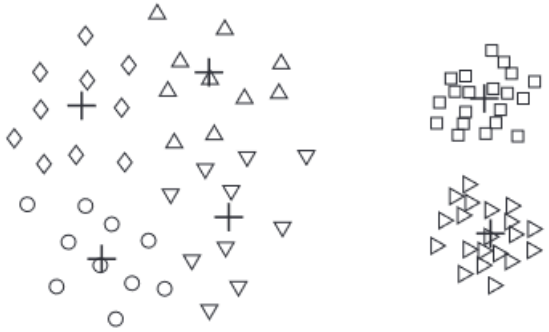
- K-means is simple and can be used for a wide variety of data types.
- K-means is quite efficient, even though multiple runs are often performed.
- Some variants (such as bisecting K-means) are even more efficient, and are less susceptible to initialisation problems.
- K-means is not suitable for all types of data, however. It cannot handle non-spherical clusters, clusters of different sizes or clusters of different densities. These limitations can often be overcome by finding pure subclusters if the number of clusters specified is large enough.
- K-means also has trouble clustering data that contains outliers. Outlier detection and removal can help significantly in such situations.
- K-means is restricted to data for which there is a notion of a centre (a centroid). A related technique known as K-medoid clustering does not have this restriction, but it is more expensive.

4.6 Cluster Evaluation

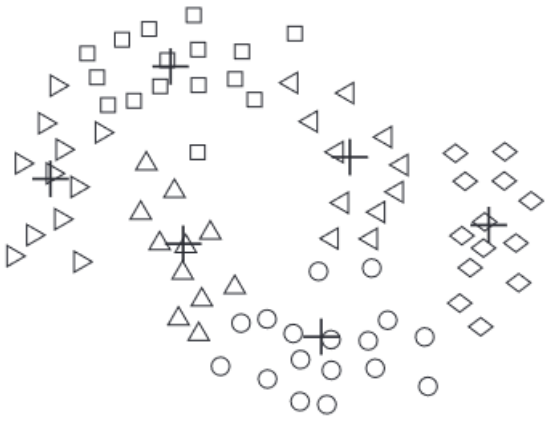
As we saw in classification (supervised), the evaluation of the resulting classification model is an integral part of the process. We also saw well-accepted evaluation measures and procedures, such as accuracy and cross-validation, respectively. However, by its very nature, cluster evaluation is not a well-developed or commonly used part of cluster analysis. After all, many times cluster analysis is conducted as part of exploratory data analysis. As such, evaluations seems to be an unnecessary complication. Added to this, there are problems arising from the different types of clusters. In some sense, each clustering algorithm defines its own type of cluster, and this may require its own evaluation measure. For example, K-means clusters might be evaluated in terms of the SSE, but for density-based clusters (which need not be spherical), SSE would not work well.

Nonetheless, cluster evaluation (traditionally known as *cluster validation*) can be important, and ought to be a key part of the cluster analysis. A key motivation here is that almost every clustering algorithm will find clusters in a data set, regardless of whether the data set actually has a natural cluster structure. This, of course, is just one consideration. There are many more, which we now outline.

(a) Unequal sizes.



(b) Unequal densities.



(c) Non-spherical shapes.

Figure 4.9: Using K-means to find clusters that are subclusters of the natural clusters. [2]

- Unsupervised techniques, i.e. do not make use of any external information.
    1. Determining the *clustering tendency* of a set of data.
    2. Determining the correct number of clusters.
    3. Evaluating how well the results of a cluster analysis fit the data *without* reference to external information.
- Supervised techniques, i.e. require external information.
    4. Comparing the results of a cluster analysis to externally known results, such as externally provided class labels.
- Can be either supervised or unsupervised.
    5. Comparing two sets of clusters to determine which is better.

For these notes, we will discuss 1, 2 and 4 only. The remaining two are left for the reader. Another point to keep in mind is that for 3, 4 and 5, we can choose to either evaluate the entire clustering, or just individual clusters. Which we choose is, as ever, situation dependent.

- Clustering tendency.

A naive approach to determining whether a given data set has clusters is to simply try and cluster it. However, most clustering algorithms will always find clusters, regardless of whether the data is highly clustered, uniformly distributed or entirely random. To overcome this, we have several options. First, we could attempt to evaluate the resulting clusters in some way and then only claim a data set has clusters if at least some of the clusters are of good quality. However, a problem immediately presents itself as clusters in the data can be of a different type than those sought by our clustering algorithms. To address this, we could use multiple algorithms and again evaluate the quality of the resulting clusters. If the clusters are uniformly poor, then this may indeed indicate that there are no clusters in the data.

Alternatively, we could attempt to evaluate clustering tendency *without* clustering. The most common approach, especially for data in Euclidean space, has been to use statistical tests for spatial randomness. As might be expected, choosing the correct model, estimating the parameters and evaluating the statistical significance of the hypothesis that the data is non-random is a challenge. Nevertheless, several approaches have been developed, especially for data in low-dimensional Euclidean space.

One example is the *Hopkins Statistic*. The idea is to generate p points that are randomly distributed across the entire space (denote the collection of these by Y) and also sample p actual data points from our data set X (without replacement), which we label Z. Usually, $p << n$, where n is the number of data points in our data set X. Then, for both sets of points, we find the distance to the nearest neighbour in our original data set. We label these as follows:

- u_i, the distance of $y_i \in Y$ from its nearest neighbour in X;

Figure 4.10: 100 uniformly distributed points. [2]

- w_i, the distance of $z_i \in Z$ from its nearest neighbour in X.

Given this, the Hopkins Statistic H is given by,

$$H = \frac{\sum_{i=1}^{p} u_i}{\sum_{i=1}^{p}(u_i + w_i)}.$$

- If the randomly generated points and the sample of data points have roughly the same nearest neighbour distances, then H will be near 0.5.
- If there are clusters, then H approaches 1 with a value of (or close to) 1 meaning the data is highly clustered.
- If the data data is uniformly distributed, then H will tend to 0.

Of course, due to sampling, the results may vary with different executions. This means we usually repeat multiple times to get an accurate measure.

Example 4.7. *Consider the data set in Figure 4.10 containing 100 uniformly distributed points. For $p = 20$ and with 100 different trials, the average value of H was found to be 0.56 with a standard deviation of 0.03. For completeness, in Figure 4.11, this data set has been clustered using K-means, where $k = 3$.*

Example 4.8. *Now consider the data set in Figure 4.12 which is obviously well separated into three clusters. In this case (for $p = 20$ as before, and where 100 trials were once again performed), the average value of H was 0.95 with a standard deviation of 0.006.*

- Determining the correct number of clusters.

There are various unsupervised cluster evaluation measures to approximately determine the 'correct' number of clusters. One such approach is to plot SSE against the number of clusters for a (bisecting) K-means clustering of the data set. For example, in Figure 4.13, there are naturally 10 clusters. In Figure 4.14 we show the aforementioned plot. We notice that there is a distinct levelling-off in the SSE when the number of clusters is equal to 10.

Figure 4.11: 100 uniformly distributed points - clustered with K-means ($k = 3$). [2]

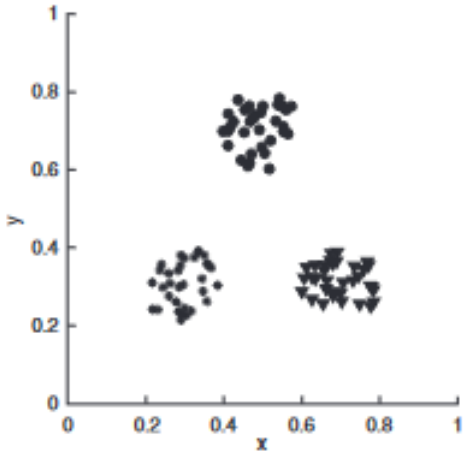


Figure 4.12: 100 uniformly distributed points. [2]

Figure 4.13: Points arranged in ten clusters. [2]

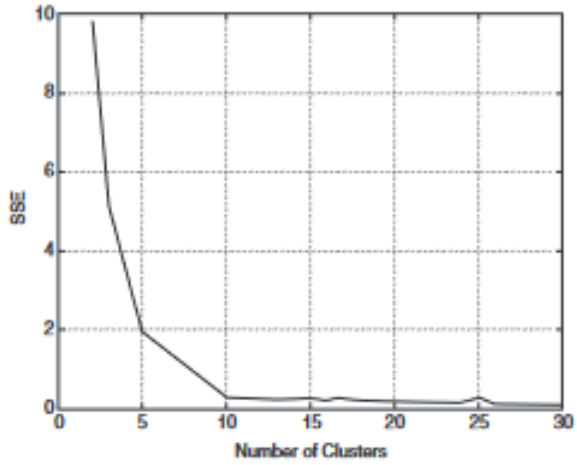


Figure 4.14: SSE versus number of clusters. [2]

- Supervised measures of cluster validity

When we have external information about the data, it is typically in the form of externally derived class labels for the data objects. In such cases, the usual procedure is to measure the degree of correspondence between the cluster labels and the class labels.

It is natural to ask why we do this since, at first glance, it appears off to perform a cluster analysis when we already have the class labels. Some motivations for this include comparing clustering techniques with the 'truth', or evaluating the extent to which a manual classification process can be automatically produced by cluster analysis. Another motivation might be to evaluate whether objects in the same cluster tend to have the same label for semi-supervised learning techniques.

We consider two kinds of approaches. The first set use measures from classification, such as entropy, to evaluate the extent to which a cluster contains objects of a single class. The second group of methods is related to the similarity measures for binary data, such as the Jaccard measure we saw earlier. These approaches measure the extent to which two objects that are in the same class are in the same cluster, and vice versa. For convenience, we refer to these two types of measures as *classification-oriented* and *similarity-oriented*.

We start with entropy and recall the definition, this time in the context of clustering. Here, the degree to which each cluster consists of objects of a single class is represented by its entropy. To compute this, we first calculate the class distribution of the data. In other words, for a cluster i we compute p_{ij}, the probability that a member of a cluster i belongs to class j as $p_{ij} = \frac{m_{ij}}{m_i}$, where

- m_i = the number of objects in cluster i;
- m_{ij} = the number of objects of class j in cluster i.

Once computed, the entropy formula is then the same as usual:

$$e_i = -\sum_{j=1}^{L} p_{ij} \log_2(p_{ij}),$$

where L = the number of classes.

The total entropy for a set of clusters is calculated as the sum of entropies of each cluster weighted by the size of each cluster, i.e.

$$e = \sum_{i=1}^{k} \frac{m_i}{m} e_i,$$

where k is the number of clusters and m is the total number of data points.

Two other performance evaluators are as follows:

1. Purity

   This measures the extent to which a cluster contains objects of a single class. With notation as above, the purity of a cluster i is given by,

   $$purity(i) = max_j(p_{ij}).$$

   The overall purity of a clustering is then given by,

   $$\sum_{i=1}^{k} \frac{m_i}{m} purity(i).$$

2. Precision

   This is the fraction of a cluster that consists of objects of a specified class. The precision of cluster i with respect to class j is $precision(i, j) = p_{ij}$.

There are, of course, other ways to measure the performance of a classification model (e.g. *recall* and *F-measure*. These are left to the reader.

Example 4.9. *We use the k-means algorithm with the cosine similarity measure to cluster 3204 newspaper articles from the Los Angeles Times. These articles come from six different classes: Entertainment, Financial, Foreign, Metro, National, and Sports. We summarise the results of the k-means clustering where* $k = 6$ *in the following table:*

Cluster	*Entertainment*	*Financial*	*Foreign*	*Metro*	*National*	*Sports*	*Entropy*	*Purity*
1	*3*	*5*	*40*	*506*	*96*	*27*	*1.2270*	*0.7474*
2	*4*	*7*	*280*	*29*	*39*	*2*	*1.1472*	*0.7756*
3	*1*	*1*	*1*	*7*	*4*	*671*	*0.1813*	*0.9796*
4	*10*	*162*	*3*	*119*	*73*	*2*	*1.7487*	*0.4390*
5	*331*	*22*	*5*	*70*	*13*	*23*	*1.3976*	*0.7134*
6	*5*	*358*	*12*	*212*	*48*	*13*	*1.5523*	*0.5525*
Total	*354*	*555*	*341*	*943*	*273*	*738*	*1.1450*	*0.7203*

Note that the first column denotes the cluster, while the next six columns indicate how the documents of each category are distributed among the clusters. Finally, the last two columns denote the entropy and purity of each cluster, respectively.

To demonstrate the computations, we consider the first cluster only. For this, $m_i = 677$. *By labelling columns 2-7 by 1-6, we see that* $m_{11} = 3$, $m_{12} = 5$, $m_{13} = 40$, $m_{14} = 506$, $m_{15} = 96$, $m_{16} = 27$. *Thus,* $p_{11} = 3/677$, $p_{12} = 5/677$, $p_{13} = 40/677$, $p_{14} = 506/677$, $p_{15} = 96/677$, $p1_{16} = 27/677$. *Putting this all together, we get the following calculations:*

$$\begin{aligned} e_1 = & \quad -\sum_{j=1}^{6} p_{1j} \log_2(p_{1j}) & = 1.22699, \\ purity(1) = & \quad max_j(p_{1j}) & = 506/677. \end{aligned}$$

Of course, the ideal situation is one in which each cluster is pure (i.e. contains only documents from one class). In reality, this is unlikely to be the case. Nevertheless, many clusters are 'nearly' pure[1]*, e.g. in Cluster 3 is very good, containing almost entirely documents from the Sports section. This is reflected by the entropy being very low/purity being very high. To improve the entropy/purity of the others, we could increase the value of* k.

Moving on, we next consider similarity-oriented measures of cluster validity. All measures are based on the premise that any two objects that are in the same cluster should be in the same class, and vice versa. We can view this approach as involving the comparison of two matrices:

1. Ideal Cluster Similarity Matrix

   This has 1 in the ij^{th} position if two objects i and j are in the same cluster, and 0 otherwise;

2. Class Similarity Matrix

   This is defined with respect to class labels and has a 1 in the ij^{th} entry if two objects i and j belong to the same class, and 0 otherwise.

N.B. We can take the correlation of these two matrices as a measure of cluster validity. This measure is known as *Hubert's* Γ *Statistic.*

Example 4.10. *Suppose the case where we have five data points* P_1, P_2, P_3, P_4, P_5, *two clusters* $C_1 = \{P_1, P_2, P_4\}$ *and* $C_2 = \{P_4, P_5\}$, *and two classes* $L_1 = \{P_1, P_2\}$ *and* $L_2 = \{P_3, P_4, P_5\}$. *We can then compute the Ideal Cluster Similarity Matrix and Class Similarity Matrix as follows:*

$$ICSM = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \quad , \quad CSM = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

In general, we can use apply any of the measures for binary similarity to these matrices (we can convert the matrices into binary vectors by appending the rows). For clarity, we repeat the definitions of the four quantities used to define those similarity measures, modifying

[1] Perhaps 'reasonably' is a better word here.

them slightly to reflect the current context. Note that there are $\frac{m(m-1)}{2}$ pairs, where m is the number of objects.

$f_{00} =$ *number of pairs of objects having a different class and a different cluster;*
$f_{01} =$ *number of pairs of objects having a different class and the same cluster;*
$f_{10} =$ *number of pairs of objects having the same class and a different cluster;*
$f_{11} =$ *number of pairs of objects having the same class and cluster.*

In particular, we consider the Simple Matching Coefficient (known as the Rand statistic in this context) and the Jaccard coefficient. We recall their definitions here:

$$Rand\,Statistic = \frac{f_{00}+f_{11}}{f_{00}+f_{01}+f_{10}+f_{11}},$$
$$Jaccard\,Coefficient = \frac{f_{11}}{f_{01}+f_{10}+f_{11}}.$$

Example 4.11. *Consider the matrices computed in the previous example. With the above notation, we have* $f_{00} = 4$, $f_{01} = 2$, $f_{10} = 2$, $f_{11} = 2$. *Thus, the Rand statistic is* $(2+4)/10 = 0.6$ *and the Jaccard coefficient is* $2/(2+2+2) = 0.33$.

Appendix A

Linear Regression

The goal here will be to explain a way linear algebra can be used in linear regression. The key is what we call a *projection matrix.* This can be written as

$$P = A(A^T A)^{-1} A^T,$$

where A is some $m \times n$ matrix.

Question: What does this matrix do?

Answer: It produces a projection of a vector $\mathbf{b}$ to the nearest vector in the column space of A.

So, if $\mathbf{b}$ is already in the projection matrix, Pb should equal $\mathbf{b}$, whereas if $\mathbf{b}$ is orthogonal to the column space (i.e. is at a right angle), then $P\mathbf{b}$ should be $\mathbf{0}$. We can check this is the case.

- Suppose $\mathbf{b}$ is in the the column space of A, i.e. $\mathbf{b} = A\mathbf{x}$, for some vector $\mathbf{x}$. Then,

$$\begin{aligned} P\mathbf{b} = &\quad A(A^T A)^{-1} A^T A\mathbf{x} \\ = &\quad A(A^T A)^{-1}(A^T A)\mathbf{x} \\ = &\quad\quad AI_n\mathbf{x} \\ = &\quad\quad\quad A\mathbf{x} \qquad\qquad = \mathbf{b}. \end{aligned}$$

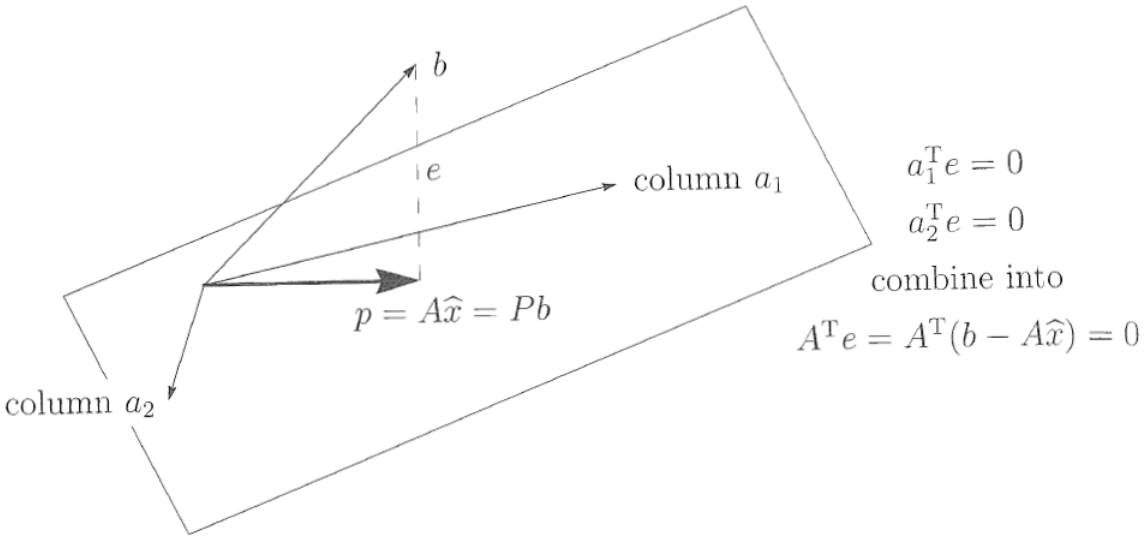
- Suppose now that $\mathbf{b}$ is orthogonal to the column space of A. So, $\mathbf{b}^T(A\mathbf{x}) = 0$ for any $\mathbf{x}$. Now, the right hand side is just a number (so it is equal to its transpose). This means the left hand side is a number (so it is equal to its transpose). Therefore, $\mathbf{x}^T A^T \mathbf{b} = 0$. This is true for any $\mathbf{x}$ and so $A^T\mathbf{b} = \mathbf{0}$. In fat, from general linear algebra, a big result says that the nullspace of A^T is the orthogonal complement of the column space of A i.e. $A^T\mathbf{b} = \mathbf{0}$. In any case, we therefore have,

$$P\mathbf{b} = A(A^T A)^{-1} A^T b = A(A^T A)^{-1}\mathbf{0} = \mathbf{0}.$$

We therefore know what happens to the extreme cases. Of course, a typical vector has some components in the column space of A, and some components orthogonal to the column

space. What the projection does is kill the orthogonal bit and preserves the part in the column space.

Geometric Picture: Suppose A is a 3×2 matrix., i.e. two columns.[1]



We have a typical vector $\mathbf{b}$ and we are projecting onto the column space. Of course, we also have another piece $\mathbf{e}$ which is the error (the bit which is not in the column space, that is being mapped to the nullspace of A^T). In particular, if $p = P\mathbf{b} = A\hat{\mathbf{x}}$ is the projected bit, then $p + e = \mathbf{b}$. In other words, the projected bit and the error together make up all of $\mathbf{b}$.

Question: We have seen that $\mathbf{e}$ is in the nullspace of A^T. So, we can think of $\mathbf{b}$ as being projected onto $\mathbf{e}$ by some matrix, in the same way that $\mathbf{b}$ is being projected onto $\mathbf{p}$ by P. What is this second projection matrix?

Answer: $I_m - P$, we just want to get the rest of the vector.

So, P and $I_m - P$ are both projections. Some other really nice facts are that if P is symmetric, then $I_n - P$ is symmetric, and if $P^2 = P$, then $(I_m - P)^2 = (I_m - P)$. If we have the above picture in our mind, this should be very natural.

Hopefully the above is not too bad. We have a vector $\mathbf{b} \in \mathbb{R}^m$, say. We have an $m \times n$ matrix A and form the matrix $P = A(A^TA)^{-1}A^T$. What this does is project $\mathbf{b}$ down into $Col(A)$ with the rest (the error) being projected onto $Null(A^T)$ by $I_m - P$. We then have $\mathbf{p} = P\mathbf{b}$ and $\mathbf{e} = (I_m - P)\mathbf{b}$. Now let's use this to perform linear regression. We will use an example of Strang (see Chapter 3 of [9]) in which we have three points $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$ given by,

$$\mathbf{b}_1 = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \ , \ \mathbf{b}_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \ , \ \mathbf{b}_3 = \begin{pmatrix} 2 \\ 3 \end{pmatrix}.$$

Problem: Find the straight line that passes through these which minimises the error between each point. See Figure A.1.

[1] Image from Strang's Linear Algebra and its Applications [9].

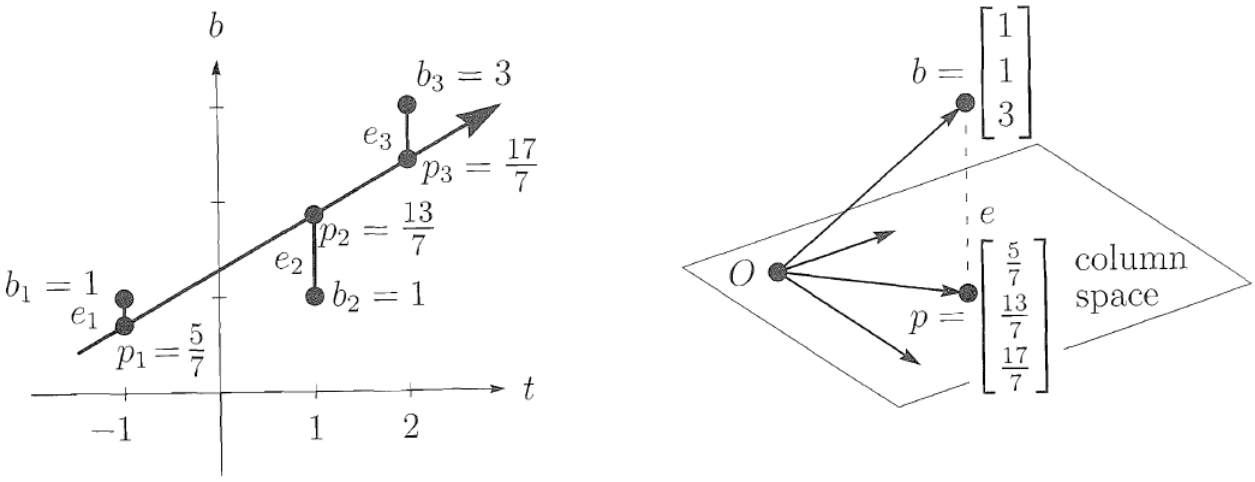


Figure A.1: Straight-line approximation matches the projection $\mathbf{p}$ of $\mathbf{b}$.

We want to find a straight line $y = c + dt$ which is the best straight line fit of these three points. Obviously, no straight line will work exactly because they do not all lie on a straight line, but we can find the best straight line.

To begin, suppose we *do* have a straight line that passes through $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$. We can then solve $1 = c - d$ (we have just plugged in the coordinates for $\mathbf{b}_1$). Likewise, we could solve $1 = c + d$ and $3 = c + 2d$. This should be recognisable. It is simply a system of three equations in two variables. We can write this as a matrix equation as follows:

$$\begin{pmatrix} 1 & -1 \\ 1 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix}.$$

Call this 3×2 matrix A and the right hand side $\mathbf{b}$. This has no solution (check this) but it has a best solution. By best solution, we mean we can minimise the error between this straight line and the initial points.

Question: How do we measure the error? What is the error?

Answer: The error is simply $A\mathbf{x} - \mathbf{b} = \mathbf{e}$, where $\mathbf{x} \in \mathbb{R}^2$ is our solution which we want to minimise. To measure the error, we can use Euclidean distance, i.e. we sum the squares and take the square root. Suppose,

$$A\mathbf{x} - \mathbf{b} = \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix}.$$

Then we want to minimise $e_1^2 + e - 2^2 + e_3^2$. Really, we should take the square root, but this doesn't change much and so it is actually convenient to minimise the square of the distance. We can now make some observations:

- The error will be $\mathbf{0}$ precisely when $A\mathbf{x} = \mathbf{b}$, i.e. when a solution exists.

- Since we are squaring, this is very sensitive to outliers. As such, we do have the usual worry when dealing with such approaches. While this method is commonly used, there are alternatives.
- There are points on this line we are trying to find. We can call them p_1, p_2, p_3 and they can be found from $A\hat{\mathbf{x}} = \mathbf{p}$, where the hat signifies that this is the best fit. In the language of linear algebra, $\mathbf{p}$ is in the columns space of A.

From the last observation, we want to find $\hat{\mathbf{x}}$ and hence $\mathbf{p}$. Since $A\hat{\mathbf{x}} = \mathbf{p}$, we know $A^TA\hat{\mathbf{x}} = A^T\mathbf{p}$ and $\mathbf{p} = \mathbf{b} - \mathbf{e}$. However, as we have seen, $\mathbf{e}$ is in the null space of A^T. Thus, we want to solve $A^TA\hat{\mathbf{x}} = A^T\mathbf{b}$. This we can do. First, we compute,

$$A^TA = \begin{pmatrix} 3 & 2 \\ 2 & 6 \end{pmatrix}.$$

This is invertible, and so $\hat{\mathbf{x}} = (A^TA)^{-1}A^T\mathbf{b}$. It therefore follows that $\mathbf{p} = A\hat{\mathbf{x}} = A(A^TA)^{-1}A\mathbf{b}$ and we are back to our projection matrix. Plugging in all the numbers, we get

$$A\hat{\mathbf{x}} = \frac{1}{14}\begin{pmatrix} 1 & -1 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}\begin{pmatrix} 6 & -2 \\ -2 & 3 \end{pmatrix}\begin{pmatrix} 1 & 1 & 1 \\ -1 & 1 & 2 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix} = \frac{1}{7}\begin{pmatrix} 5 \\ 13 \\ 17 \end{pmatrix}.$$

Note that if we went back to $A^TA\hat{\mathbf{x}} = A^T\mathbf{b}$ then we could solve this without finding inverses and so on. If we multiply everything out then we are actually solving,

$$\begin{array}{rcl} 3\hat{c} + 2\hat{d} & = & 5 \\ 2\hat{c} + 6\hat{d} & = & 6 \end{array}.$$

These are called the *normal equations*. Further, if we went back to the problem of minimising $e_1^2 + e_2^2 + e_3^2$, where $e_1 = (c - d - 1)$ and so on, then we could approach this from a calculus point of view. In other words, expand out the squares, then differentiate with respect to c, d and set to 0. This will once again give the above normal equations. We leave this as an exercise.

A.0.1 Weighted Least Squares

A simple least squares problem is the estimate $\hat{x}$ of a patient's weight from two observations $x = b_1$ and $x = b_2$. Unless $b_1 = b_2$, we have an inconsistent system of two equations in one unknown:

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} x = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

Until now, we have accepted b_1, b_2 as equally reliable. We have looked for the value $\hat{x}$ that minimised $E^2 = (x - b_1)^2 + (x - b_2)^2$, i.e. $\frac{dE^2}{dx} = 0$ at $\hat{x} = \frac{b_1 + b_2}{2}$. The optimal $\hat{x}$ is the average. The same conclusion comes from $A^TA\hat{x} = A^Tb$. In fact, A^TA is a 1×1 matrix and the normal equation is simply $2\hat{x} = b_1 + b_2$.

However, it is very possible that the two observations are not to be trusted to the same degree. Perhaps b_1 is obtained from a more accurate scale, or has been measured by a

professional, whereas b_2 is given by the patient. Perhaps b_1 is obtained from a larger sample and so is more trustworthy. Nevertheless, if b_2 contains some information, we do not want to rely totally on b_1. The simplest compromise is to attach different weights w_1^2 and w_2^2 and choose $\hat{x}_W$ to be the solution which minimises the *weighted sum of squares:*

$$\textbf{Weighted error} \qquad E^2 = w_1^2(x-b_1)^2 + w_2(x-b_2)^2.$$

If $w_i > w_j$, we attach more importance to b_i over b_j. The minimising process (derivative $= 0$) tries harder to make $(x-b_1)^2$ small:

$$\frac{dE^2}{dx} = 2[w_1^2(x-b_1) + w_2^2(x-b_2)] = 0 \text{ at } \hat{x}_W = \frac{w_1^2 b_1 + w_2^2 b_2}{w_1^2 + w_2^2}.$$

This time, instead of the average of b_1 and b_2, $\hat{x}_W$ is a *weighted average* of the data. If $w_1 > w_2$, the average is closer to b_1 than to b_2.

The ordinary least-squares problem leading to $\hat{x}_W$ comes from changing $A\mathbf{x} = \mathbf{b}$ to the new system $WA\mathbf{x} = W\mathbf{b}$. This changes the solution from $\hat{\mathbf{x}}$ to $\hat{\mathbf{x}}_W$. The matrix W^TW then turns up on both sides of the weighted normal equations. Specifically, the least squares solution to $WA\mathbf{x} = W\mathbf{b}$ is $\hat{\mathbf{x}}_W$ and the weighted normal equations are

$$(A^TW^TWA)\hat{\mathbf{x}}_W = A^TW^TW\mathbf{b}.$$

Question: What happens to our picture of $\mathbf{b}$ being projected to $A\hat{\mathbf{x}}$?

Answer: The projection $A\hat{\mathbf{x}}_W$ is still the point in the columns space that is closest to $\mathbf{b}$, but now the word 'closest' has a new meaning. This time, we use the *weighted length*, i.e. instead of orthogonal meaning $\mathbf{y}^T\mathbf{x} = 0$, it means $(W\mathbf{y})^T(W\mathbf{x}) = 0$. Really, what we are doing is working with a new inner product. Before we used the Euclidean inner product, now we are using the weighted Euclidean inner product. The only hiccup is if W is non-invertible, so we avoid this and assume W is invertible.

In practice, the important question is the choice of W^TW. The best answer comes from Gauss. We may know that the average error is zero (that is, the expected value of the error in $\mathbf{b}$, although the error is not expected to be 0). We may also know the average of the square of the error (that is, the variance). If the errors in the b_i are independent of each other, and their variances are σ_i^2, then the 'right' weights are $w_i = \frac{1}{\sigma_i}$. Here, a more accurate measurement, which means smaller variance, gets a heavier weight.

In addition to unequal reliability, the observations may not be independent. The poll for PM are not independent of those for the party as a whole. In such cases, the errors are coupled and W has off-diagonal terms. The best unbiased matrix W^TW is the inverse of the covariance matrix. Then the main diagonal of $(W^TW)^{-1}$ contains the variances of σ_i^2, which are the average of $(error\ in\ b_i)^2$.

Appendix B

SQL

Bibliography

[1] E.F. Codd, S.B. Codd, and C.T. Smalley. *Providing OLAP to User-Analysts: An IT Mandate*. E.F. COdd and Associates, 1993.

[2] P. Tan et al. *Introduction to Data Mining*. Global Edition. Pearson, Second edition, 2020.

[3] S. Musa et al. Using persistent homology as preprocessing of early warning signals for critical transition in flood. *Scientific Reports*, 11, 2021.

[4] J.E. Evans. Why logarithms to the Base e Can Justly Be Called Natural Logarithms. *National Mathematics Magazine*, 14(2):91–95, 1939.

[5] R. Ghrist and Y. Hiraoka. Applications of Sheaf Cohomology and Exact Sequences on Network Codings. *Proc. NOLTA*, 2011.

[6] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. MK Publishers, third edition edition, 2012.

[7] A. et al. Holzinger. *On Entropy-Based Data Mining. In: Holzinger, A., Jurisica, I. (eds) Interactive Knowledge Discovery and Data Mining in Biomedical Informatics. Lecture Notes in Computer Science*, volume 8401. Springer, Berlin, Heidelberg, 2014.

[8] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, third ed. edition, 2002.

[9] G. Strang. *Linear Algebra and its Applications*. Cengage Learning, Boston, fourth ed. edition, 2005.

[10] Xindong W. and Xingquan Z. Mining With Noise Knowledge: Error-Aware Data Mining. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 38(4):917–932, 2008.