ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

APPLIED DATA ANALYSIS

School of Computer and Communication Sciences

Applied Data Analysis
Lecture Notes

Dr. Catasta Michele
Distributed Information Systems Laboratory (LSIR)
michele.catasta@epfl.ch

December, 2016

Contents

Course content

The purpose of the course is to present multiple topics in the data science field such as *Data Wrangling*, *Data Management*, *Data Mining*, *Machine Learning*, *Visualization*, *Statistics* and *Story telling*. The course has not the presumption to go deeply into each argument. It is due to the extent of the subject and the fact that it is evolving really quickly, hence learning in depth a specific tool will not pay off.

Skills to develop

In this course, you will work, thus develop, the following skills:

Data mining/scraping/sampling/cleaning in order to get an informative, manageable data setlength

Data storage and management in order to be able to access data quickly and reliably during subsequent analysis

Exploratory data analysis to generate hypotheses and intuition about the data

Prediction based on statistical tools such as regression, classification, and clustering

Communication of results through visualization, stories and interpretable summaries

Structure of the notes

The notes of the lectures are put in writing with the aim of summarizing the main topics and concepts illustrated during the classes. To those who are curious, it points to external links that may help you to an in-depth understanding of the field. Each chapter corresponds to a lecture. The hope is that the work could be useful for the current and future students.

1 Introduction

1.1 What is Data Science?

When we talk about Data Science, we often use the term Big Data as the enormous amount of data that exists in the world. But Big Data is not only about collecting huge amount of data. It is challenging but not enough. The real value comes from the insights. The *internet* companies (Google, Facebook, etc.) understood this many years ago.

An accurate definition of Data Analysis is given by Wikipedia:

> **Analysis of data** *is a process of* **inspecting**, **cleaning**, **transforming**, *and* **modeling data** *with the goal of* **discovering useful information**, *suggesting conclusions, and supporting decision-making. Data analysis has multiple facets and approaches, encompassing diverse techniques under a variety of names,* **in different business**, *science, and social science* **domains**. Wikipedia - Data Analysis

Therefore, a Data Scientist has to master different kinds of skills such as **Mathematics** , **Statistics**, **Programming** and the **Domain Expertise**. Drew Conway's Venn diagram, Figure 1.1, shows the different combination man can obtain with these three skills.

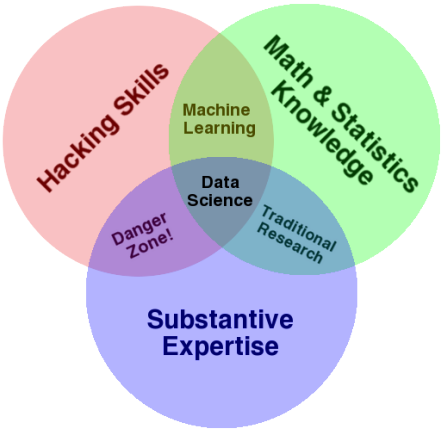


Figure 1.1: Venn Diagram describing the different combination of skills used by a Data Scientist (by Drew Conway)

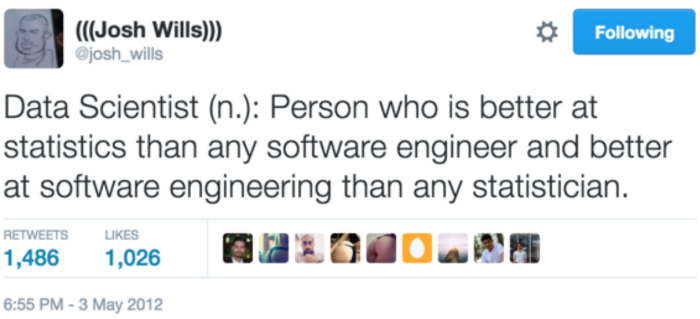
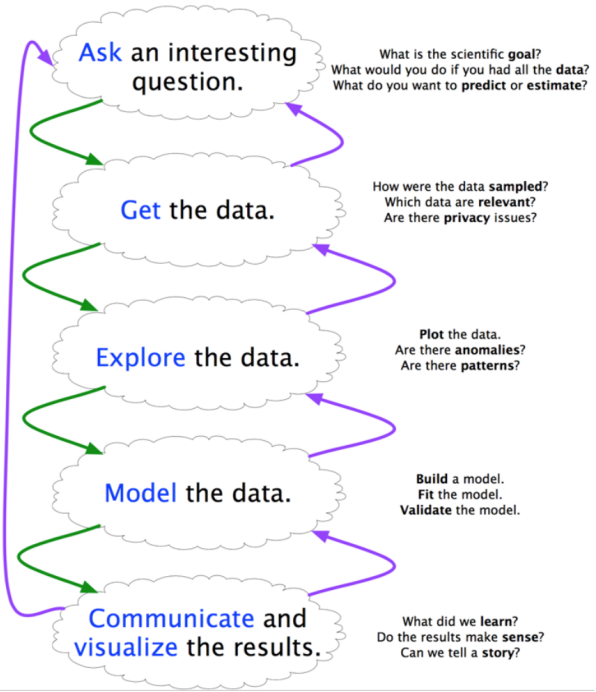


Figure 1.2: A tweet from Josh Wills, Data Scientist at Slack.

Whether you are interest in knowing what other people think about *Data Science* there are many ongoing discussions.

A more practical definition

Data Science is about the whole processing pipeline to extract information out of data. As such, a Data Scientist **understands and cares about the whole data pipeline**.



A **data pipeline** consists of 3 steps:

1. Preparing to run a model: *Gathering, cleaning, integrating, restructuring, transforming, loading, filtering, deleting, combining, merging, verifying, extracting, shaping*
2. Running the model
3. Communicating the results

A "good" Data Scientist will always go back and forth between the steps. The diagram on the left shows exactly what can happen.

2 Basic concepts

2.1 A taste of "grammar"

A data science student is attended to understand the *Grammar of Data Science*. We are going to learn the core data manipulations, so no matter what is the backend is `SQL`, `pandas` and `R`(with `dplyr`), all share the same *grammar*. For sure having some backgrounds in SQL concepts is a good thing because, as it is very common, people love to make examples with it. Here is a brief refresh of some definitions and concepts that are essential and we need to be aware of.
Let's start with two key concepts that *Structured data* requires:

- **Data model**: a collection of concepts for describing data.
- **Schema**: a description of a particular collection of data, using a given data model.

The most common and ubiquitous approach to manage data is the *Relational model* (E. F., Ted Codd). It can handle most of the data, so most of it is "reduced" to this model. To have an idea, a counter example is the facebook-like data which requires **graph model**.
The *Relational model* is composed by relations. A *relation* is made up two parts:

1. The **Schema** specifies name of relation, plus name and type of each column.
   For example, `Students(sid: string, name:string, age:integer)`
2. The **Instance**, *i.e.* the data at a given time.
   Definitions:
   - **Cardinality** is the number of rows (= number of items)
   - **Degree** or **Arity** is the number of fields (= number of attributes)

Database vocabulary

Here follows a list of basic words and operations that should be kept in mind when we talk about *Databases*.

- A `JOIN` is a mean to combine tables based on shared attributes (most of the time some `IDs`). Despite its apparent simplicity beware of the many ways to compute a `JOIN` and check what is the default `JOIN` of a language before using it. Every different `JOIN` operation has implications on the resulting relation. The Figure 2.1 summarises these possibilities.
- *Aggregation*, *reduction*, and `groupby` are the actions of reducing data with a common operation (`sum`, `count`, `average`, ...) to summarize them. Remember that you always need to specify the *attribute* you are going to perform the *aggregation* on.

Whether you are not too familiar with *Database* you can go through this course and look up for what you need.

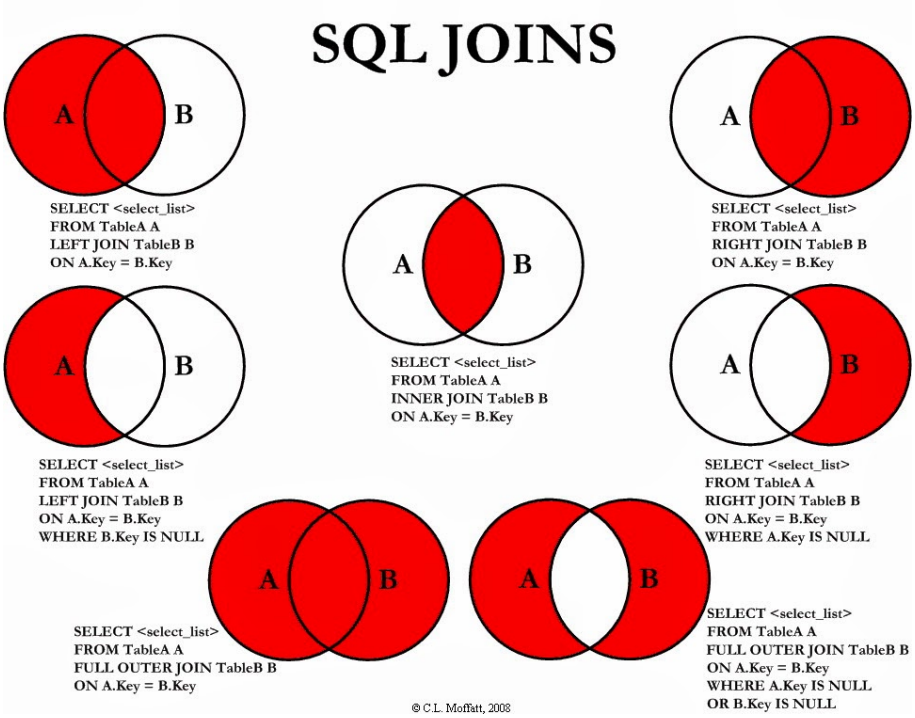


Figure 2.1: Different ways to join two tables and the related SQL command. Wikipedia - Join (SQL)

2.2 Our tools

During this course and, likely, in your future, to handle data you are going to use a *'magic'* `Python`'s tool that has a black and white coat, with black fur around its eyes: `pandas`. The basic ingredients of our beloved `pandas`:

- `Series` are a name, ordered dictionary
  - keys are indexes
  - built on `numpy.ndarray` (so values can be any `Numpy` data type)
- `DataFrame` is a table with named column
  - the columns are series
  - it is indeed a dictionary with (columnName → series)

Learn how to use `pandas` and discover the beauty of `IPython Notebooks`.

2.3 Panda vs SQL

`pandas` is built to allow easy and fast *data exploration* and not to be a database manager, as SQL is. Thus there are benefits and drawbacks of using it.

Pros	**Cons**
+ Lightweight & fast	- Tables stored directly in memory
+ Great expressiveness (combine SQL + Python)	- No post-load indexing functionality
+ Easy plot for data visualization (e.g. Matplotlib)	- No transactions, journalings
	- Large, complex joins are slower

2.4 OnLine Analytical Processing (OLAP cubes)

OLAP tools enable users to analyze multidimensional data interactively from multiple perspectives. Conceptually, it is like an n-dimensional spreadsheet (a cube) on which we can apply various operations to take decisions.
OLAP cubes are another way to see data tables and are constructed based on them, as shows Figure 2.2.

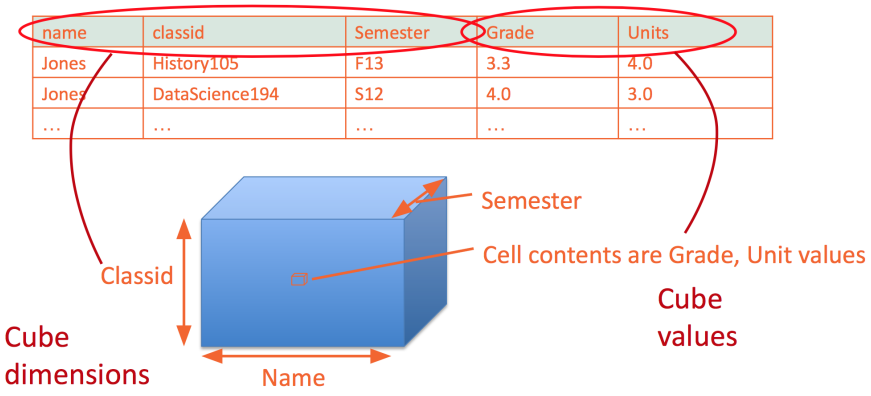
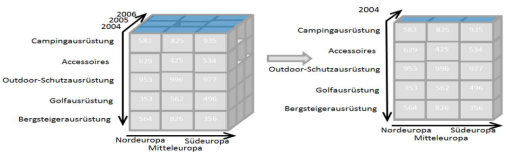


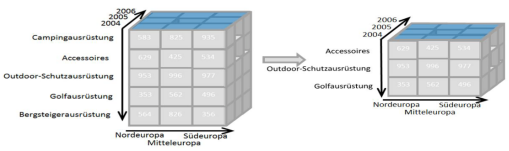
Figure 2.2: Construction of an OLAP cube from a table.

Operations on OLAP cubes are the following and are illustrated on Figure 2.3

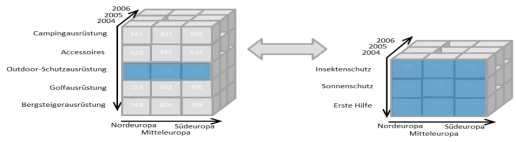
- *Slicing* fixes one or more variable
- *Dicing* selects a range of one or more variable
- *Drilling up/down* change levels of a hierarchically indexed variable, i.e. "zoom" on a variable and see the subcategories it contains.
- *Pivoting* change the point of view of the cube. Swap an aggregated variable and a detailed one.



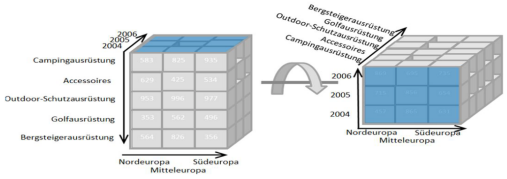
(a) Slincing



(b) Dicing



(c) Driling up/down



(d) Pivoting

Figure 2.3: Operations on OLAP cubes

Pros	**Cons**
+ The main advantage of OLAP cubes is that they are **conceptually simpler** to understand by a non-scientist person, e.g. a business man who has to take day-to-day decisions based on company's data. Aggregations are limited but cover the main common cases that we can encounter.	- Because of the "on-line" behaviour of this approach, all types of aggregation must be pre-calculated among all combinations of axes which is very **expensive in memory and in time** (when updating the data)

3 Data Wrangling

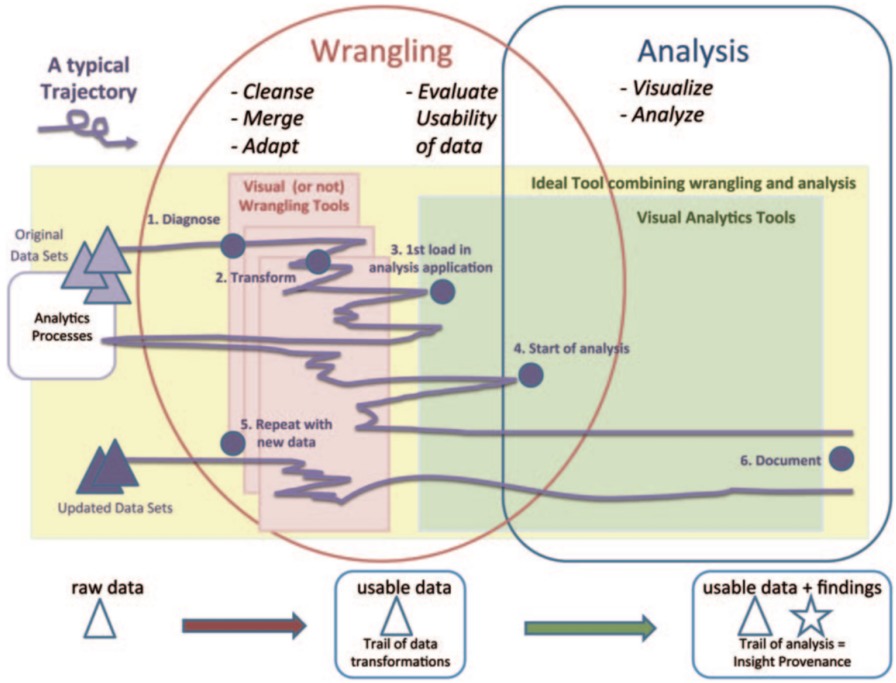


Figure 3.1: Things do not always happen as expected...

Before any analysis, data need to be transformed from "dirty" to clean and processable data.
Data come from different sources, sometime collected through different methods over time, with different conventions. Generally, it leads to unwanted anomalies and duplicates. *Data wrangling* goal is to **extract and standardize these raw data**. The best way to do it is to **combine** automation with visualizations in order to proceed with the cleaning.
Here is a *non-exhaustive* list of where data problems can come from:

- *Missing data*
- *Incorrect data*
- *Inconsistent representations of the same data*
- *Non-standardized data (centimeter or inches? Fahrenheit or Celsuis ?)*
- *Duplicated data*

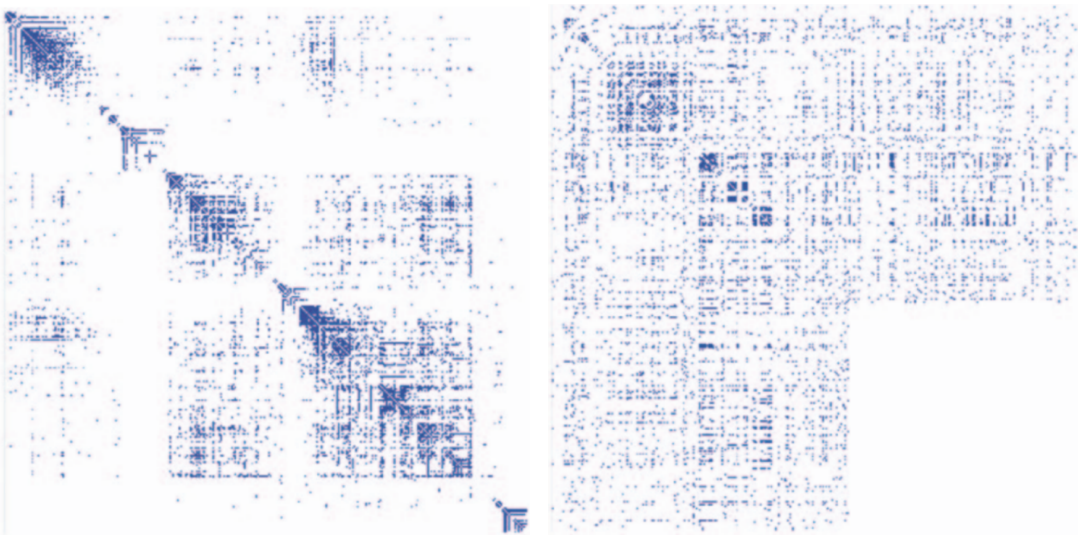
About 75% of theses problems will need **human intervention** to be corrected (by the data-scientist or by crowdsourcing). Some examples of good data leading to horrible conclusion can be found here: Dirty Data Horror Stories.
Even if it seems really dirty, **beware not to over-sanitize the data!** Applying what we can call "defensive programming" is not a good idea because we risk losing any interesting data, keeping only the ones that fit perfectly in our model.
Anyway, a good *Data wrangling* procedure brings about improvements in the efficiency and scale of data importing.

3.1 Diagnosis of the data

One of the most important aspects of Data Wrangling is to **understand** the data and to **find** possible problems. In order to "diagnose" the data, two tools can be used:

- *V*isualization (A *thoughtful* visualization will always help)
- *B*asic Statistics

Look out, outliers and missing data can be often identify using a plot. Visualization becomes increasingly difficult when data gets larger.
Matrix visualizations of the facebook graph is shown in Figure 3.2. The Relational visualization, Figure 2(a), does not show any particular problem in the data. But the Time dependant visualization, Figure 2(b), shows that the Facebook API reached its limit while collecting data.



(a) Relational visualization (b) Time dependant visualization

Figure 3.2: Matrix visualization of the facebook graph.

3.2 Dealing with missing values

Values can often miss from the data we have, because of various events (war, fire, ...). We must detect and correct these values with different methods according to the domain we are working in.
Whatever the method used, it's good to keep track of these changes to know which are original data and which are modified ones.

- Set values to zero Figure 3.3(a)
- Interpolate based on existing data Figure 3.3(b)
- Omit missing data Figure 3.3(c)
- Interpolation with tracks kept Figure 3.3(d)

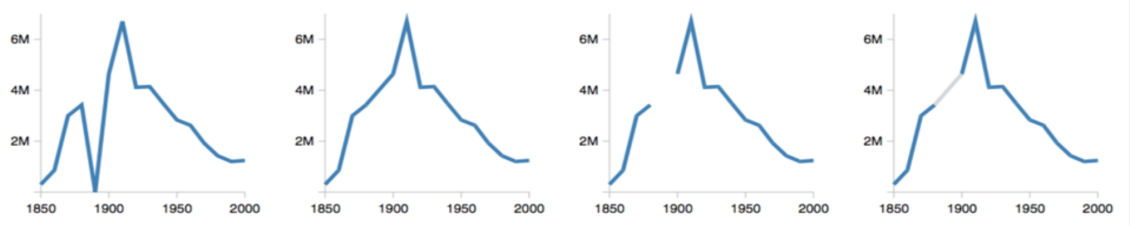


Figure 3.3: To deal with missing values.

3.3 General procedure

Once the data are well wrangled and before trying to analyze them, we must take care of two more steps:

1. **Deal with uncertain data** (can arise from measurement errors, wrong sampling strategies, etc.)
2. **Parse/transform data** (with aggregation and reduction techniques) to obtain meaningful records

As we say in the Introduction, *Data Science* is a mixture of different fields. It often leads to the necessity of working in team where different people have different skills. Working in a team means sharing code, documentation and data. Hence, taking care of them is essential.

4 Data Variety

The "3 Vs" of Big Data: *Volume*, *Velocity* and *Variety*. In this course, we don't address the *Volume* and *Velocity* parts (A course on Database does). Since there is a lot of variety in the data, we need to prepare the data. This flow is called **ETL**:

- **Extract** from the *source(s)*.
- **Transform** data at the source, sink, or in a *staging area*.
- **Load** data into the *sink*.

This variety of the data comes, in a first place, from the many different sources from which we extract them: *files, databases, logs, ...* Each of these sources uses (or not!) its proper convention and can contain structured (DB), semi-structured (logs) or unstructured (web page) data.

4.1 Role of Schema

The **Schema**, which specifies the *structure* and *types* of data repository, is changing. Traditional databases are **schema-on-write**, *i.e.* you cannot load data into a table without a schema. But new data stores (NoSQL for example) are **schema-on-read** or **schemaless**.

- **Schema-on-write** is typically SQL, where we must create a table before inserting data in our system. Data must scale the defined schema and this is both the strength and weakness of the system. Strength because the data is perfectly oriented and respect the constraints we establish. Weakness because schemas are always subjective in some ways and data (which are perfectly correct) may not fit with it.
- **Schema-on-read** is for instance XML, where you create the schema according to the data you read.
- Youtube and Google Cache where the first **schemaless** data system. Without schema, everything is simply stored as a string and we need a parser to return a typed data.

4.2 Examples of data

4.2.1 XML and DOM

The XML data are used mostly with HTML and specifies the data structure. An XML schema can be applied to interpret the XML data and specifies the **data types**. Figure 4.1 shows the XML data and Figure 4.2 shows the schema used to parse and type the data.

```
<location>
  <latitude>37.78333</latitude>
  <longitude>122.4167</longitude>
</location>
```

Figure 4.1: XML data defined by the schema in Figure 4.2.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="unqualified">
<xsd:complexType name="location">
  <xsd:sequence>
    <xsd:element name="latitude" type="xsd:decimal"/>
    <xsd:element name="longitude" type="xsd:decimal"/>
  </xsd:sequence>
</xsd:complexType name="location">
```

Figure 4.2: XML schema for the XML data in Figure 4.1.

The XML is a text format that encodes **DOM** (Document-Object Models). It's a data structure often used by Web pages. The DOM is tree-structured. An example of a DOM is given in Figure 4.3.

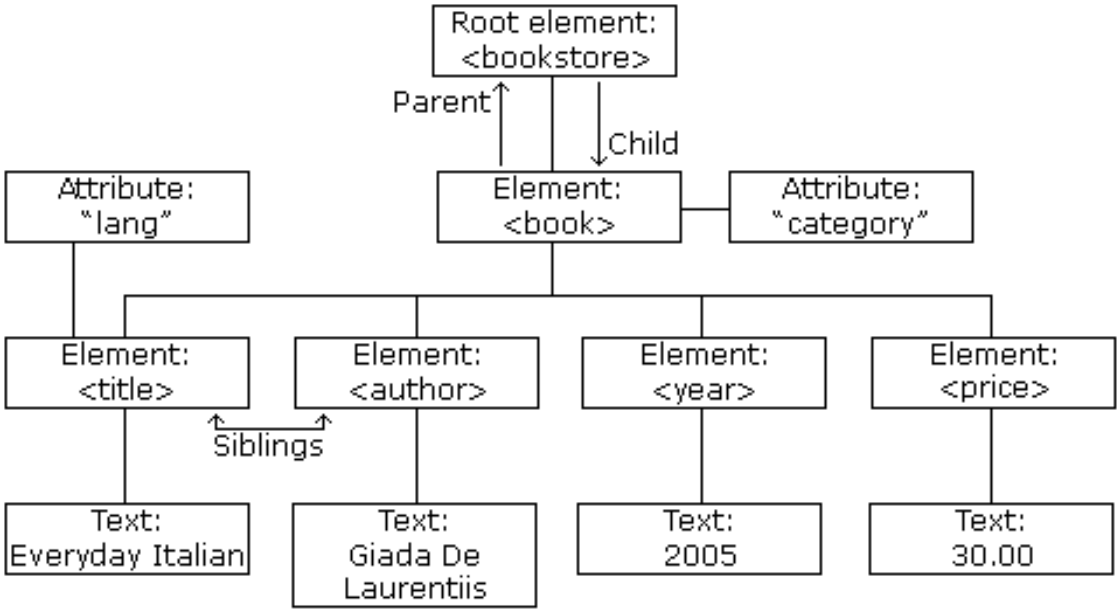


Figure 4.3: Example of a DOM tree for an HTML Web page. See example on W3Schools.

THe XML schema allows a database to interpret the data when running queries. It can do arithmetic or range queries on numerical values, for example.

4.2.2 JSON

JSON stands for Javascript Object Notation. It's a schemaless data (schema support was added later). An example of JSON data is shown in Figure 4.4

JSON is typically used to represent **hierarchical data structures** directly in the target language (Javascript or Java at the beginning). The transformation on the data is procedural in the target languages. It is often easier for some tasks, but it can be painful for some of them: for example schema changes.

```json
{
  "firstName": "John",
  "lastName": "smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

Figure 4.4: Example of a JSON data.

4.2.3 Tabular data

A Tabular Data is simply data put into a table such as CSV (Comma Separated Value) or TSV (Tab Separated Value). Definition of a table:

- A **table** is a collection of **rows** and **columns**.
- Each row has an **index**.
- Each column has a **name**.
- A **cell** is specified by an (index, name) pair.
- A cell may or may not have a **value**.

It's a very simple yet powerful data type. For example, the sensors usually output data in the form of time series, transformed into a tabular format. However, a system dealing with sensor data should:

- support both long-term (**trend**) and short-term (**real-time**) queries
- have **low latency** but also efficient. It should use **real-time indexing** for longer-term queries.
- support triggers (**alerts**) for a variety of conditions.

Therefore, the **complexity of a data format** does not determine the **complexity of the system required to properly handle it**.

4.2.4 Log files

The log files are simple text files giving information about the process. The daemons, such as `httpd`, `mysqld` or `syslogd`, usually create logs. `syslog` was developed by Eric Allman. It's a way for devices to send event messages to a server that will log all the events. Splunk is a company that built a successful business model around the syslog events.

4.2.5 Binary formats

They are often the key to performance because we **avoid expensive parsing**. The modern formats even support nested structures, various levels of schema enforcement, **compression**, etc. Some examples: Protocol Buffers (Google), Avro (Apache), Parquet (Apache), etc.

4.3 Processing the data (JSON and XML)

In order to process XML, we can use the DOM. It can also be used to process JSON data. The DOM is very easy to work with: all the data are directly accessible by links. The problem is that we **might not care about most of the data** and if the data are big, they **might not fit into the RAM**. In order to deal with these two problems, we can use a **SAX** parser which is an event-driven parser. It will find all the **open-close-tag events** in an XML document and will bf do callbacks to user code.

+ User code can respond to only a subset of events corresponding to the tag it is interested in.
+ User code can correctly compute aggregates from the data rather than create a record for each tag.
+ User code can implement flexible error recovery strategies for ill-formed XML.
– User code must implement a state machine to keep track of "where it is" in the DOM tree.

For JSON, most parsers construct the "DOM" directly. But there are a few SAX-style parsers: Jackson, JSON-simple, etc. Sometimes **SAX-style is the way to handle ill-formed datasets**, an endless array of objects, for example.

4.4 HTML and Web Services

4.4.1 HTML

Internet contains an "enormous" amount of data. Some crawlers such as Common Crawl datasets contains about 1.82 billion web pages (for 145 TB). We can use different tools to crawl data from the web. Examples for Python: Beautiful Soup, Requests, Scrapy, etc.

Most of the time, the Web pages are considered as unstructured data. But you can find some semi-structured data, *e.g.* Google WebTables. Some big "internet" companies (Google, Yahoo, Yandex and Microsoft) are sponsoring a project called schema.org to create structured or semi-structured Web pages. A core vocabulary for the type of fields is given. schema.org is more and more used. It's also used by knowledge bases such as Google Knowledge Graph. WikiData is a community project to create an open database of structured data taken from Wikipedia.

4.4.2 Web Services

Screen-scraping the content of a large website was possible before, but become more and more difficult nowadays. This is mainly due to the content "hidden" behind a form or an authentification. Take for example facebook without account, or the IS-academia course page if you do not select a semester. Therefore big companies are providing Web Service APIs[1] to retrieve data from their website. There are two kinds of Web Services:

- The old way: XML-based RPC-style messages: SOAP
- The new way: REST-style stateless interactions, URLs encode state

4.4.2.1 RPC

The SOAP RPC [2] messages typically encode arguments that are presented to the calling program as parameters and return values. HTTP POST/GET are used to communicate.

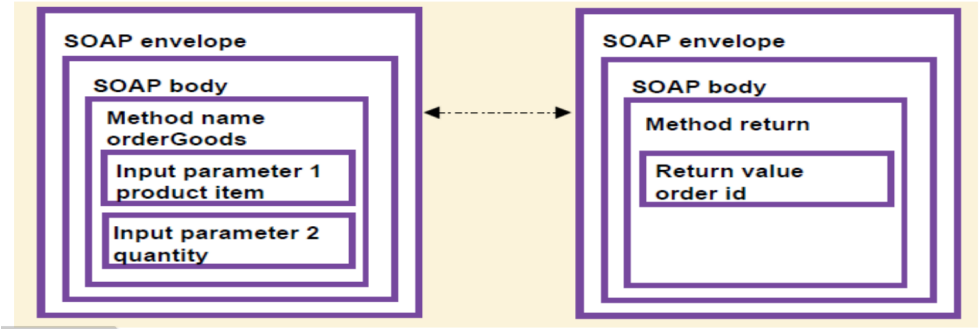


Figure 4.5: Example of a SOAP RPC exchange.

This kind of procedure (same for XML-RPC) requires a request-response cycle. This often leads to longer "conversations". The RPC-style is being quickly superseded by newer and more user-friendly technologies.

In **RPC systems**, the design emphasis is on **verbs**. It uses functions such as *getUser()*, *addUser()*, etc.

4.4.2.2 REST

REST[3] is a **stateless** client/server protocol. The principles are:

1. Each message in the protocol contains all the information needed by the receiver to understand and/or process it. This constraint attempts to *"keep things simple"* and avoids needless complexity.
2. Set of Uniquely Addressable Resources
   - *"Everything is a Resource"* in a RESTful system

[1] Application Program Interface: Set of subroutine definitions, protocols, and tools for building software and applications. In this particular case, the APIs are used to retrieve the data from the Web page, *e.g.* Facebook API to retrieve the contacts.
[2] SOAP = Simple Object Access Protocol, RPC = Remote Procedure Call
[3] REpresentation State Transfer

- Requires universal syntax for resource identification, *e.g.* URI.

3. Set of Well-Defined Operations that can be applied to all resources

   - In the context of HTTP (REST APIs), the primary methods are:
     POST, **GET**, **PUT**, and **DELETE**
     These are similar (but not exactly) to the database notion of CRUD (Create, Read, Update, and Delete)

4. The use of Hypermedia both for Application Information and State Transitions

   - Resources are typically stored in a structured data format that supports hypermedia links, such as XHTML or JSON.

In **REST systems**, the design emphasis is on **nouns**. It uses the HTTP Protocols (POST, GET, PUT, and DELETE) a *User*, a *Location*, etc.

5 Statistics on the Data

When we explore and analyze data, it would be great if we only had to look at some statistics numbers and make automatically a conclusion about them. Sadly, it's not the case. At all.

5.1 Examples of famous mistakes due to statistics

5.1.1 Anscombe's quartet: Sensivity of outliers & Robust statistics

The FIG 5.1 show four different data distribution that present, despite all of that, the same means on x and y, the same variance on x and y, and, thus, the same linear regression function. This is due to the statistics used to define them.

- Min, Max, Mean, Standard Deviation (Std) and Range are sensitive to outliers and then are **not robust statistics**.
- Median, quartils, (and others) are not sensitive and then are said to be **robust statistics**.

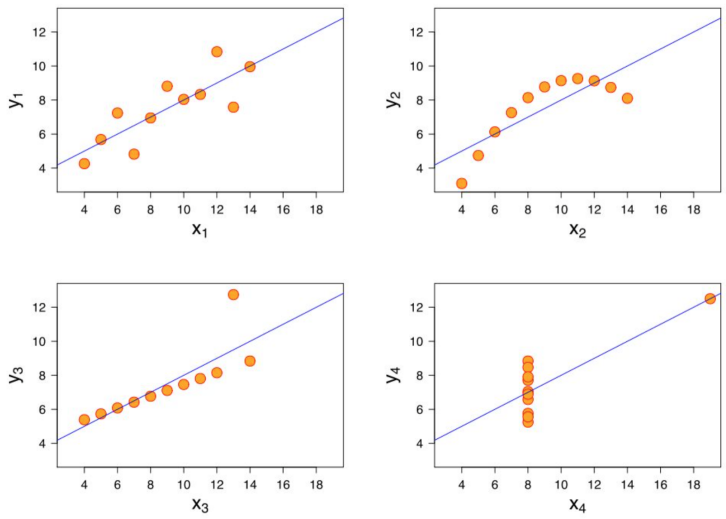


Figure 5.1: Anscombe's quartet

5.1.2 Simpson's paradox: aggregation of data

Certain tendencies can appear, disappear or even reverse themselves when aggregating the data! This was the case when media started blaming Berkey of being unfair with women applications (looking at left table of FIG 5.2). After further investigation (and de-aggregation of the data), it appeared that at the opposite... Berkeley was unfair with men! (Right table of the same FIG).
This paradox comes from the fact that women (according to these tables) tended to apply for more competitive departments, with lower rates of admission. **When aggregating the data, we lost this subtlety and then draw a wrong conclusion.**

Simpson's paradox can appear in a lot of cases and can be very hard to detect. The wikipedia page of Simpson's paradox describes a lot of examples and, for the ones interested, a great book relates lots of statistical errors that drove to miscarriages of justice: Leila Schneps and Coralie Colmez, Math on Trial: How Numbers Get Used and Abused in the Courtroom

	Applicants	Admitted
Men	8442	**44%**
Women	4321	35%

Department	Men		Women	
	Applicants	Admitted	Applicants	Admitted
A	825	62%	108	**82%**
B	560	63%	25	**68%**
C	325	**37%**	593	34%
D	417	33%	375	**35%**
E	191	**28%**	393	24%
F	373	6%	341	**7%**

Figure 5.2: Berkley admission tables of 1973

5.2 Refresh of basic statistics concept

- **Probabilities**: mathematical theory that describes uncertainty.

- **Statistics**: Set of techniques for extracting useful info from data

> **Probability and Statistics** *are related areas of mathematics which concern themselves with analyzing the relative frequency of events. Still, there are fundamental differences in the way they see the world:*
> ***Probability deals with predicting the likelihood of future events**, while **statistics involve the analysis of the frequency of past events**.*
> *Probability is primarily a theoretical branch of mathematics, which studies the consequences of mathematical definitions. Statistics is primarily an applied branch of mathematics, which tries to make sense of observations in the real world.*
> Steven S. Skiena, "Calculated Bets", Cambridge University Press, 2001

5.2.1 Bayes Theorem

The theorem express the very intuitive statement that:
The probability of observing event A and B is the probability of observing B multiplied by the probability of observing A knowing that B occurred.
Mathematically it's expressed as:

$$P(A|B)P(B) = P(A \cap B) = P(B|A)P(A) \tag{1}$$

Or equivalently

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2}$$

More about Bayes Theorem on wikipedia.

5.2.2 Random Variables

A **random variable** is a quantity that can take various values, each one associated with a probability of apparitions. The some of these probabilities will always be 1.

$$X : \Omega \to E \tag{3}$$

Ω being a probability space and E a measurable space (usually $E = \mathbb{R}$).
Any random variable can be described by its cumulative distribution function, which describes the probability that the random variable will be less than or equal to a certain value.
Two **independent variables** are defined as

$$\mathbb{P}(A \cap B) = \mathbb{P}(A) \cdot \mathbb{P}(B). \tag{4}$$

or equivalently (by Bayes Theorem)

$$\mathbb{P}(A \mid B) = \mathbb{P}(A) \tag{5}$$

More about Bayes Theorem on wikipedia.

5.2.3 Law of Large Numbers

The Law of Large Numbers links, in some way, the probability to the statistics. It's, again, a very intuitive statement, even if not so easy to prove (as always in mathematics).

> In probability theory, the **law of large numbers** (LLN) is a theorem that describes the result of performing the same experiment a large number of times. According to the law, **the average of the results obtained from a large number of trials should be close to the expected value**, and will tend to become closer as more trials are performed. Wikipedia

A common mistake is to deduce that, in the case of playing heads or tails for example, observing a lot of time **heads** increase the probability of observing **tails**. This is absolutely wrong. The variables are perfectly independent and, according to Eq 5, the probability stays exactly 50%. This is the perfect example of confusing probabilities with statistics.

5.2.4 Central Limit Theorem

Central Limit Theorem states that the mean of independent and identically-distributed random variables will converge to **Gaussian Distribution** (Normal Distribution).

5.3 Most common distributions

- **Gaussian Distribution** (fig 5.3) results from independent and identically-distributed variables $f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\sigma^2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ More on wikipedia
- **Poisson Distribution** (fig 5.4) describe the observation of events happening in a delimited time-laps. E.g: an event happens in average 4 times each 10 minutes ($\lambda = 4$). What's the probability that it appears after only 3 times in this same interval ($k = 3$)? $p(k) = \frac{\lambda^k}{k!} e^{-\lambda}$ More on wikipedia

- **Exponential Distribution** (fig 5.5) describes the time between two events in a Poisson process. $P(x) = \lambda e^{-\lambda x}$ More on wikipedia

- **Binomial Distribution** (fig 5.6) describes the discrete distribution on success in a yes/no experiment. (E.g. coin tossing or any win/lose game). $f(k; n, p) = \Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$ More on wikipedia

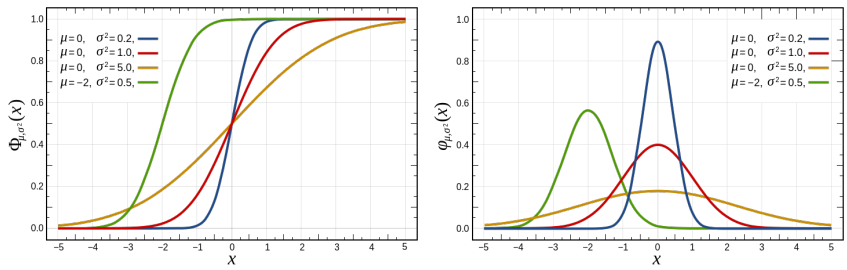
- **Multinomial Distribution** generalizes Binomial law. More on wikipedia

- **Zipf Distribution** is an empirical discret description of word frequency in a text. More on wikipedia

- **Pareto Distribution** is the equivalent of Zipf in a continuous space. It allows, amongst other things, to give a theoretical base of the "80-20 principle" (20% of the causes produce 80% of the effects). More on wikipedia

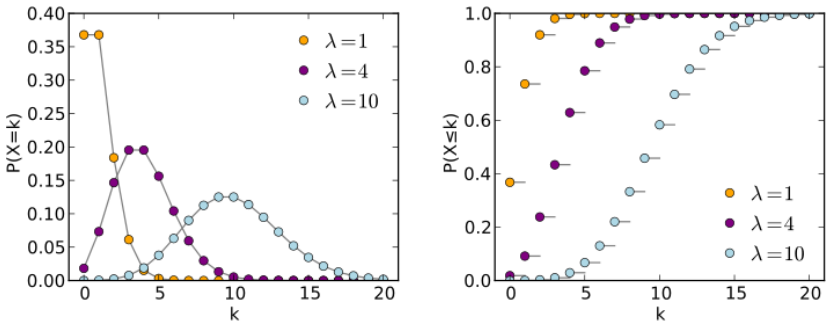
- **Yule-Simon distribution** describes discret frequencies of term too. More on wikipedia

You should understand the distribution of your data before applying a model!



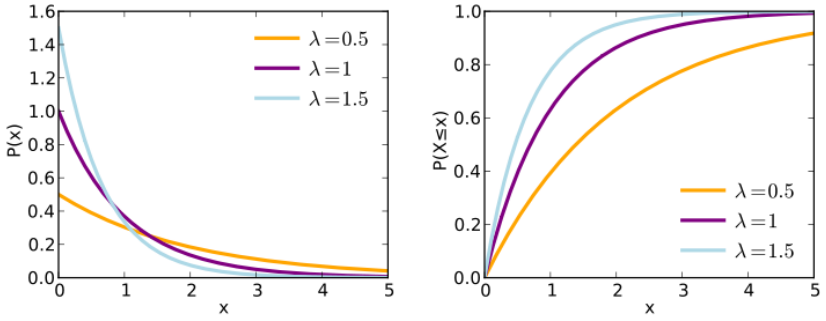
(a) Probability density function (b) Cumulative distribution function

Figure 5.3: Gaussian distribution



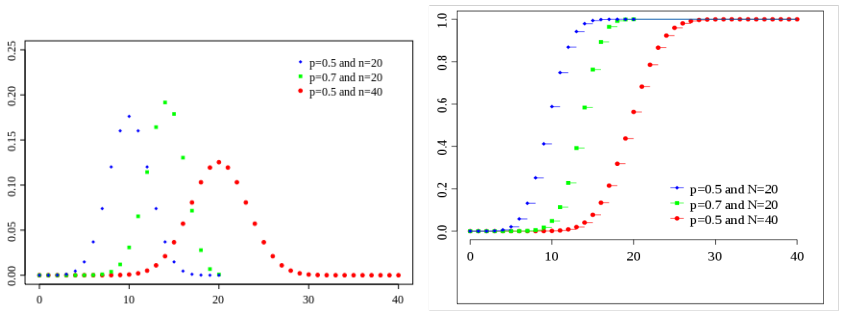
(a) Probability density function (b) Cumulative distribution function

Figure 5.4: Poisson distribution



(a) Probability density function (b) Cumulative distribution function

Figure 5.5: Exponential distribution



(a) Probability density function (b) Cumulative distribution function

Figure 5.6: Binomial distribution

5.4 Measurement on Samples

In practice, we (almost) never analyse the entire population. We always work on a subset of it called **sample**. The **variance** is the variation between elements of our sample, that we hope to be the same as the population. The **biases** is the systematic variation between the entire population and the sample we chose.
When randomly select elements of the population to be part of the sample, we have a great chance that the bias is small (i.e. that the distribution of the sample corresponds to the distribution of the population). But do not forget that there is a probability (even if a small one) to select elements that **biased our measures**! This probability can even increase when you clean the data, if you don't do it wisely.
A stupid example could be a study on population education in which, during the cleaning, you remove the answers containing misspelling. Uneducated people are more likely to commit misspelling so, removing them, you artificially biais the sample.

5.5 Test Statistic

(The only good and easy-to-understand explanation about test statistics I ever found is available on hamelg.blogspot)
The idea behind test statistics is **instead of proving that our assumption is true, let's calculate the probability that our observations occur by chance (null hy-**

pothesis or H_0). If this probability is very low, then there is a good chance that our hypothesis is true!
The probability that this happens by chance is called *pvalue* and we usually consider that if $pvalue \leq 0.05$ our hypothesis is true ($pvalue \leq 0.01$ in some strict cases).

5.5.1 Example with t-test

Let's take the example on FIG 5.7. The **Null Hypothesis** H_0 represents the distribution of observation we can do, assuming there is no correlation between the values we measured. The **Alternative** is H_A, our hypothesis which states that, at the opposite, there is correlations.

- If the observation we test is $x = 3$, there is **less than 5% probabilities that it was produced by** H_0 ($pvalue \leq 0.05$). Our hypothesis H_A is considered true.
- If the observation we test is $x = 0$, there are **more than 5% probabilities that it was produced by** H_0 (more or less 40%). Our hypothesis H_A is considered false.

An important thing to notice is that **it does not provide the truth on statistic**, it only provides information about how likely is the null hypothesis! Let's look back to the example

- The $x = 3$ observation **could have been produced by the red area**, meaning that it's part of the small 5% chance of being produced by the H_0. The test will say that our hypothesis is true, which will be a **false positive**.
- The $x = 0$ observation **could have been produced by the blue area**, meaning that even it H_0 have great chance to produce it, it was in fact produced by H_A. The test will say that our hypothesis is false, which will be a **false negative**.

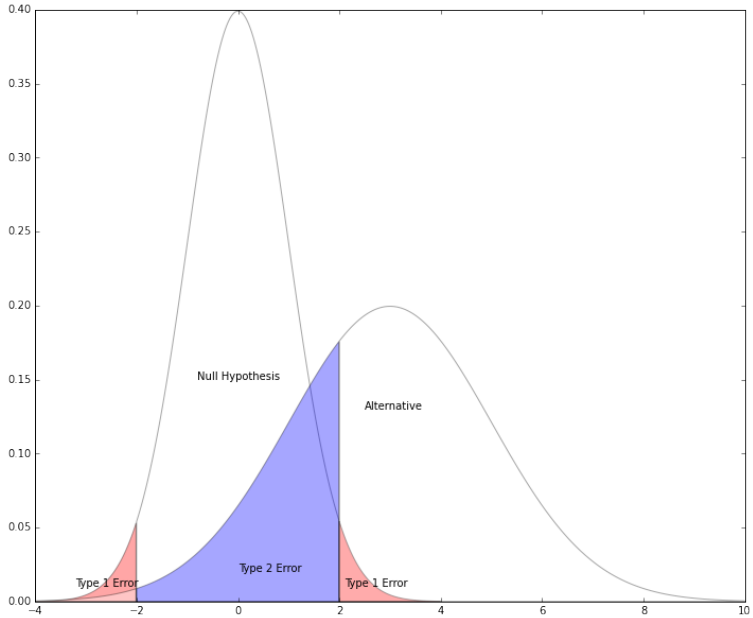


Figure 5.7: T-Test example

5.5.2 Choose the right test

A lot of tests exist and we must choose wisely which one to use, according to data and hypothesis characteristics. FIG 5.8 show a decision tree helping to choose the test which suits best our situation.

- Question ?
- Data type ?
- Sample size
- Variance known?
- Variance of several groups equals?
- ...

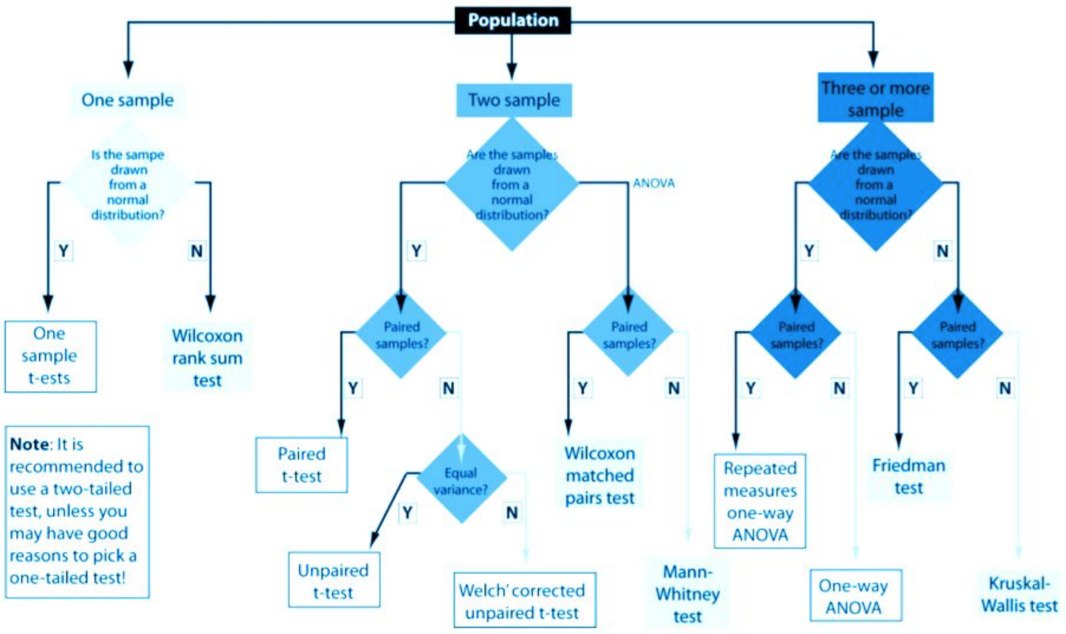


Figure 5.8: Statistical test decision tree

5.5.3 Family-wise Error

Following this simple math equation, we can figure out that the more experiment we do to test the hypothesis, the more likely we are to find that one of them are spuriously right! This is because the reverse point of view of the test. We are testing the fact that H_0 is unlikely, not directly that H_A is likely.

$$\begin{aligned} P(false\ positive) &= \alpha = 0.05 \\ P(true\ positive) &= 1 - \alpha = 0 - 95 \\ P(true\ positive\ on\ each\ experiment) &= (1-\alpha)^k \\ P(at\ least\ on\ false\ positive\ on\ one\ experiment) &= 1 - (1-\alpha)^k \end{aligned} \tag{6}$$

To counter that, two possible correction exists

- Bonferroni correction : $\alpha_c = \frac{\alpha}{k}$
- Sidak correction: $\alpha_c = 1 - (1-\alpha)^{1/k}$

5.5.4 Non-Parametric tests

All the tests so fare assume that the data are **normally distributed** and that the samples are **independent of each other and all have the same distribution.** (IID) They may be inaccurate if those assumptions are not met. Therefore, make sure the data satisfy the assumptions of the test we're using. Watch out for:

- **Outliers** will corrupt many tests that use variance estimates.
- **Correlated values as samples**, e.g. if you repeated measurements on the same subjective
- **Skewed (bias) distributions** give invalid results.

Some tests make no assumptions and thus can be used on very general cases: **K-S test**, **Permutation test** and **Bootstrap confidence interval**.

5.5.5 K-S test

K-S (Kolmogorov-Smirnov) test is a very useful test for checking whether two (continuous or discrete) distributions are the same.

- In the **one-sided test**, an observed distribution (e.g. some observed values or a histogram) is compared against a reference distribution (e.g., power-law).
- In the **two-sided test**, two observed distributions are compared.
- The K-S statistic is just the **max distance between the CDFs** (Cumulative Distribution Function) of the two distributions.
- The K-S test can be used to test **whether a data sample has a normal distribution** or not.
- Thus it can be used as a sanity check for any common parametric test (which assumes normally distributed data).
- It can also be used to compared distributions of data values in large data pipeline: **Most errors will distort the distribution of a data parameter and a K-S test can detect this.**

This test is expensive! Check for more information on the wikipedia page.

6 Data Visualization

6.1 Two main purposes

In this chapter we are going to talk about visualization in general and then about what it is called interactive visualization. The first thing we do when we talk about visualization is to split it into two different tracks.

- The first one is about *analysis*: you want to support reasoning about information. For instance, when you have a `DataFrame` you can make a plot of the distribution of the attribute in order to identify outliers, missing data and *so forth*. In general with this kind of visualization you can do more like discover structures, quantify values and influences. **This way of using visualization is extremely important for debugging purposes**.
- The second is the *communication* part and it is about informing and persuade people. The key difference in working in *Data Science* and exclusively in *Machine Learning* or *Statistics* is the fact that you don't just stop after getting a good model and evaluating its accuracy. You would make a story that you convey to people. For this reason we have to use visualization that can capture attention, can engage people and can tell a story visually (tell a story using visual tools takes a lot less time) and last but not least, you are focused only on certain aspects omitting others. It is a double-edged sword. That is because, on one hand you have to consider that there is an information overflow that people are suffering in general (we get too many media in which we consume information) so we do not want our visualization conveys more information than a human being can actually get in a few seconds). On the other hand, we have to do it carefully, avoiding omitting some information just because difficult to handle or with, apparently, nonsense.

6.2 Data exploration

In order to do a good *Data exploration* analysis by means of visualization:

- Get familiar with your favorite graphing package:
  - `Matplotlib` which is widely used in `Python`
  - `Seaborn` and `Bokeh` that are two additions on top of `Matplotlib`
  - `D3.js` (`Javascript`) is the most famous framework for interactive graphics
- Get fluent with plotting:
  - Histograms
  - Scatter plots
  - Line and bar plots

6.2.1 One variable

Whenever we want to look at the data we can use histograms, they tell us a lot about the single variable. Once you plot them, you can try to figure out their distribution, for instance we can identify skewed distributions, multimodal or long tail data (Figure 6.1).

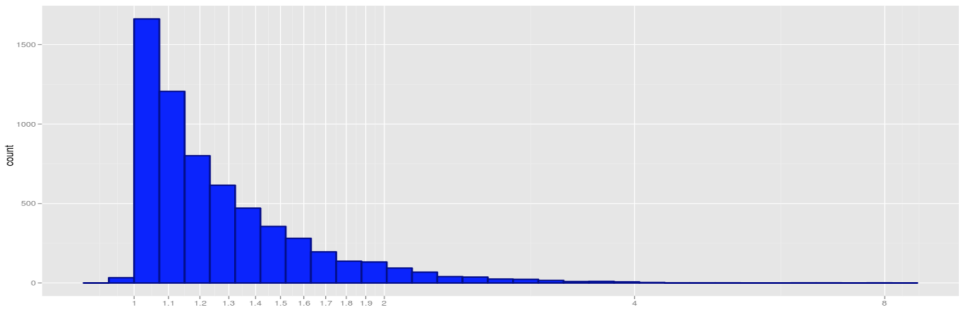


Figure 6.1: Example of long tail data.

The latter is characterized by a bunch of bins that reveal a lot of occurrences and bars in the tail where we observe a very few occurrences. Many of this long tail data follow a *power-law*. To claim the latter we need to run some test on data that proofs the statistical significance of our hypothesis, otherwise we can just state that it looks like a *power-law*. For a graphical representation:

1. Sort the histogram counts by magnitude, descending.

2. Plot count vs bucket number on a log-log plot.

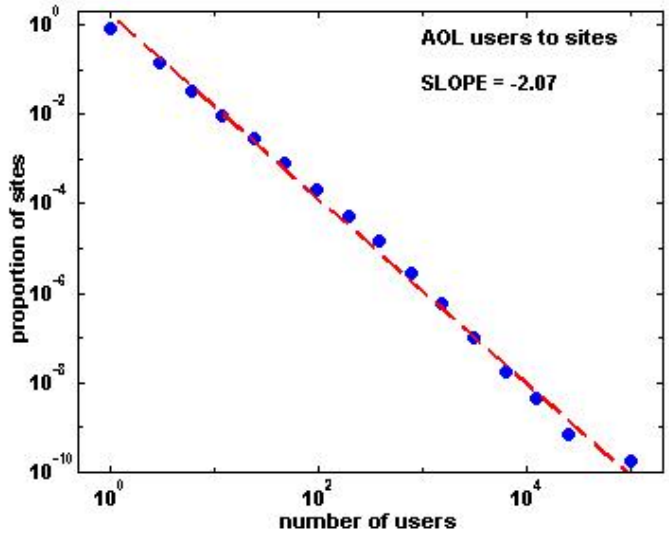


Figure 6.2: Example power law.

Generally this law is characteristic of social-influence processes, to know more look up for *Preferential attachment*.
The *multinomial* data registers more than one peak in the histogram, it suggests that there are two or more distinct populations of a sample. When you deal with something like this do not guess! Explore further by using, e.g., color and a histogram of multiple populations.

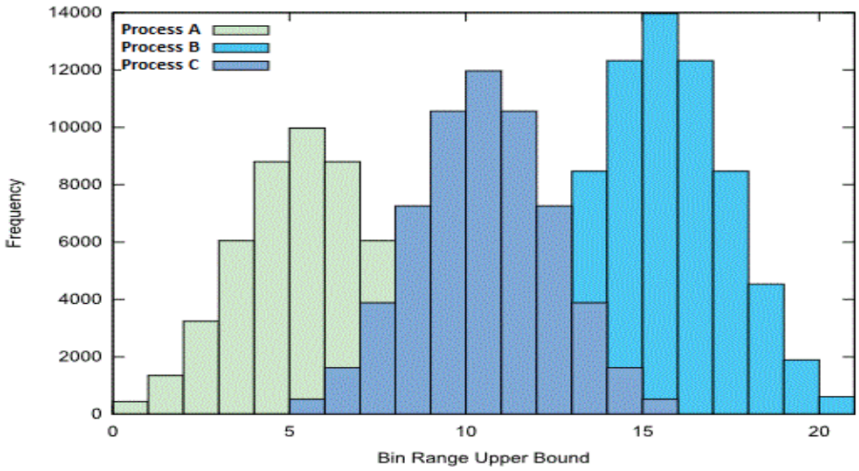


Figure 6.3: Example of multimodal data.

Sometimes data is weird and is very hard to explain. Also in this case, do not guess! Trace through the data pipeline to find where the strangeness comes from. Usually it is a processing bug. Hence, check your code!
There is a way for a *proactive Weird data Detection*. If data looks normal, take a picture and save it for later, then periodically compare new data with old whenever there is a pipeline update. Generally always try to have a theory of what the data should look like!

6.2.2 More than one variable

Most of the time we are interested in visualizing more than one variable, here a *non-thorough* list of possibilities is listed:

- Two variables
  - *Scatter plots* quickly expose the relationships between two variables
- More than two variables
  - *Stacked plot*: stack variable is discrete, useful to explore data (Figure 6.4)
  - *Parallel coordinate plot*: one discrete variable, an arbitrary number of other variables (when this number increases it risks becoming very messy), see an example in Figure 6.5
  - *Radar Chart*: one discrete variable (through the radar design), an arbitrary number of other variables (Figure 6.6)

When you deal with a high number of variables, a valid idea to visualize in a better way is to reduce the number of variables applying algorithms, this pocess is called *Dimensionality reduction*, one example is the *PCA*. Intuitively, given twenty different variables, many tend to not variate a lot, *PCA* extracts the couple of attributes that really make the difference allow visualization of high-dimensional continuous data in 2D using principal components. Hence, instead of directly plot multivariate data, try to think whether a dimensionality reduction can be useful.
We argued for analysts is important to form expectations of what the data should look like. This helps against pipeline errors and to identify interesting patterns.

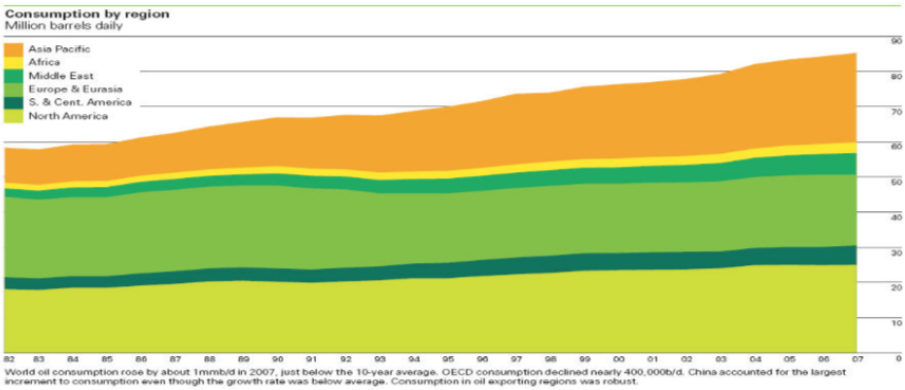


Figure 6.4: Example of stacked plot.

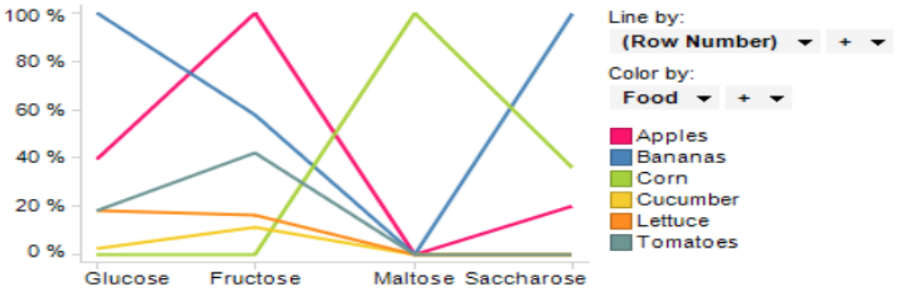


Figure 6.5: Example of parallel plot.

But beware of seeing *Martian Canals*: do not see things that are not there. Moreover, an observer should also be attuned to patterns that are not part of his theory, in other words, to expect the unexpected.

6.3 Moving Towards Interactive Viz

Interactive visualization is a new field and it's getting more and more common. Our aim is to deliver results and this has been enabled by the new web technologies and in general by few frameworks essential for the current state of the art. `JavaScript` plays a very important role in the field. The vast majority of the libraries that allow to do visualization are in `JavaScript`, so if you know how to use it or if you want to learn how to use it, it is definitely a good tool to have in your toolbar.

A visualization is worth a thousand words! Representing in 2D more than two variables has been being a challenge since a long time ago. Even without the help of the machines, someone tried to do something in this field. Nowadays the technology allows the *researchers* to go further and to use all their creativity and skills, nevertheless so many efforts should be put in. Lots of concepts have been developed and, for those interested, there are pioneers of the field that should be taken into consideration.

As we already said the visualization has the characteristic of engaging the audience easily and it often results clear and understandable (be careful, not all the graphs and repre-

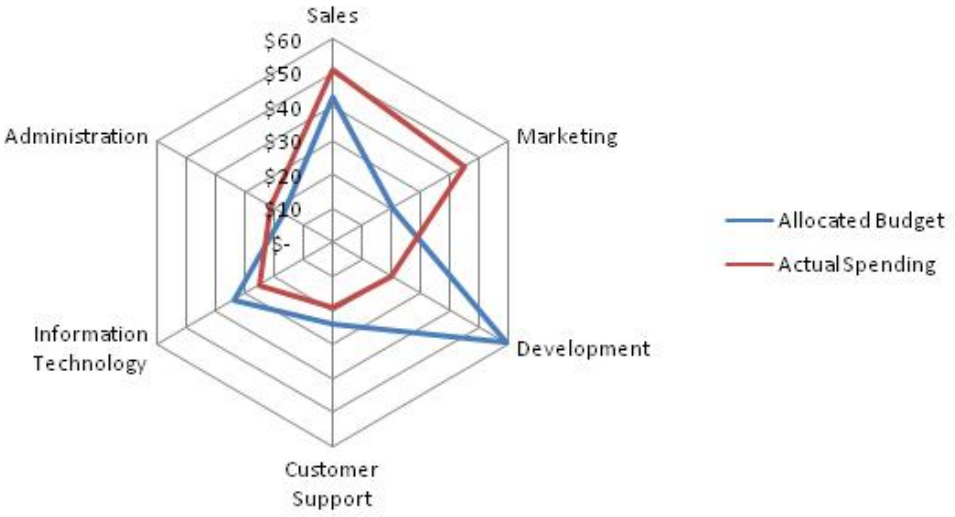


Figure 6.6: Example of radar plot.

sentations you look at are reliable, fair and proper!!). Hence it can be used to globally describe phenomena not easy to understand otherwise. Today, many are the websites where you can play with data using visualization.
Visualizing data is becoming a new way to spread information. More than ever, we recognize the existence of *Data journalism*, more data are available, many people have programming skills. Thus, it is simple to find persons who combine both writing and programming skills. There are journals that stand out, such the *New York Times*, *Forbes* and *The Economist*. They build up teams of researchers who are the best experts in the field. Hence, they can be considered a great source of best practices in viz.

6.4 Visualization definitions

There is not a unique way to define visualization, here some definitions that try to include many few aspects are listed:

- *Transformation of the symbolic into the geometric* (McCormick et al. 1987)
- *... finding the artificial memory that best supports our natural means of perception.* (Bertin 1967)
- *The use of computer-generated, interactive, visual representations of data to amplify cognition.* (Card, Mackinlay & Shneiderman 1999)

The 10 rules

When you do visualization you have to take care of the following rules:

1. **Show your data**: be careful of showing what you want to show, do not forget the main information;
2. **Use graphics**: glue together descriptions and figures;
3. **Avoid Chartjunk**: display your data in a fancy way, do not add anything that can make the interpretation harder;

4. **Utilize Data-ink** as much as you can, be careful when choosing what keep and remove;
5. **Use labels**: let the people understand what you are talking about;
6. **Utilize Micro/Macro**: an overview does not need so many details as when you zoom in;
7. **Separate Layers**: make more visible what you want the people to focus on;
8. **Use Multiples**: a thing different from the other captures the attention;
9. **Utilize Color** in a way such that data is interpreted;
10. **Understand Narrative**: when you tell a story respect time and space.

Interactive chart design

With interactive charts you can keep things very simple by hiding and dynamically revealing important structure. On an interactive chart, you reveal the information most useful for navigating the chart. The aforementioned rules hold for the interactive charts as well.

The importance of magnitude

Compare areas

Let the reader compare areas is dangerous (Figure 6.8), avoid it whether possible or try to insert information to be able of making significant comparisons. Related to the capability of distinguishing and understanding magnitudes, in 1984, Cleveland and McGill wrote the paper *Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods* which identifies and analyzes a set of *elementary perceptual tasks* conducted in the moment the reader extract quantitative information from graphs (Figure 6.7).

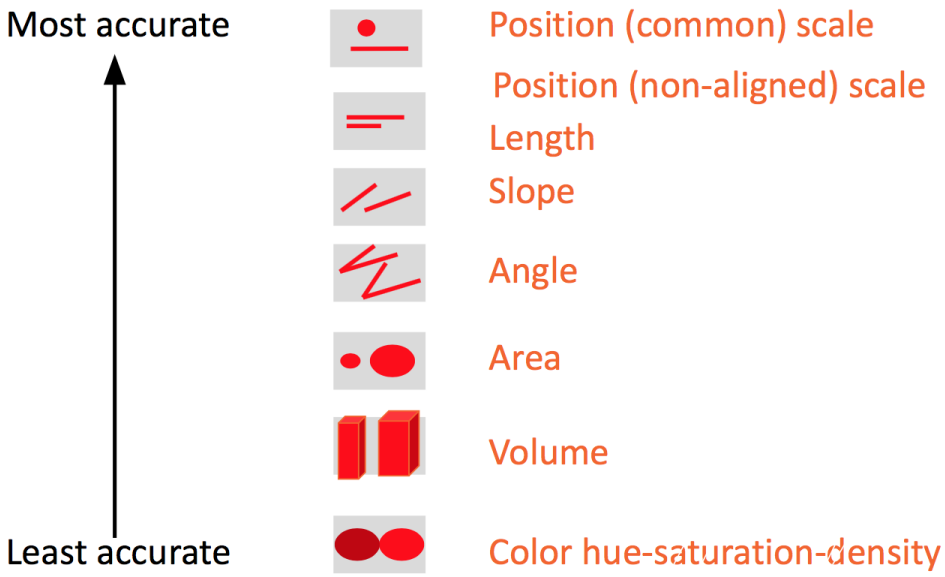


Figure 6.7: What works and what does not.

These tasks are sorted according to how accurately people perform them.

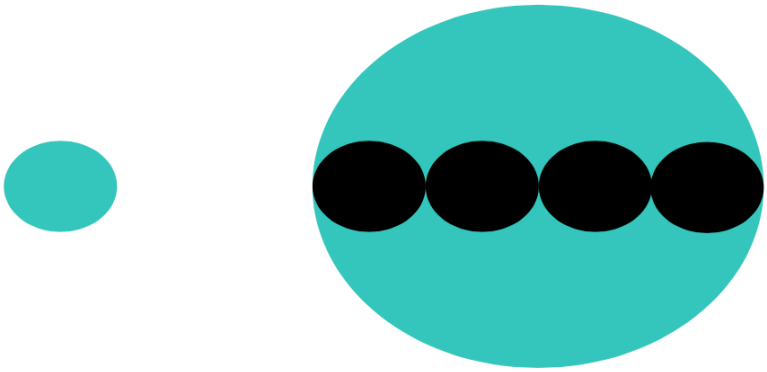


Figure 6.8: How many times the *little* one is included in the *big* one? (Answer: 16)

Compare colors...

Use a magnitude that allows people to easily read and interpret your data, noticeable differences are required. In 1846 the physicist Ernst Weber, defining I as the intensity of the stimulus and S the sensation, said that

$$\Delta S = k\frac{\Delta I}{I}.$$

It is known as the *Weber's law* and reads out that a variation in the sensation is proportional to the magnitude of the original intensity of the stimulus. So as the base I increase, we require a larger changes in ΔI to notice the change.

...And choose them

Choose colors based on the information you want to convey:

- *Sequential*: colors can be ordered from low to high (Figure 6.9)
- *Diverging*: two sequential schemes extended out from a critical midpoint value (Figure 6.10)
- *Categorical*: Lots of contrasts between each adjacent color (Figure 6.9)

Figure 6.9: Example of sequential colors.

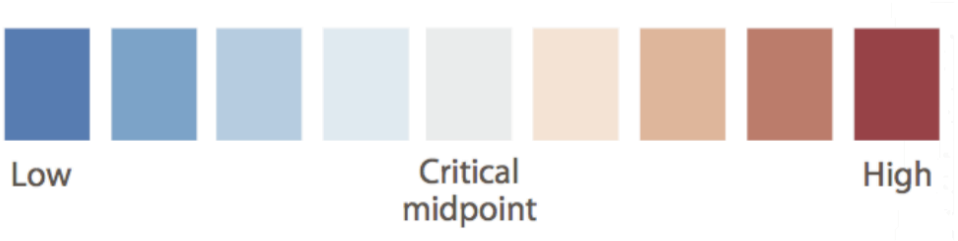


Figure 6.10: Example of diverging colors.

Figure 6.11: Example of categorical colors.

The usage of these tools depends on what you what to show. Anyway there are several online sources that can help you to choose the color scheme according to your purpose.

Use Structure

In 1912 Gestalt outlined principles that describe how our mind organizes individual visual elements into groups, to make sense of the entire visual. When designing a visual, these principles can be used to highlight patterns that are important to us, and downplay other patterns. The Figure illustrates the principles of Gestalt which is relevant to visualization (Figure 6.12). Do not concentrate too much information, less is more!

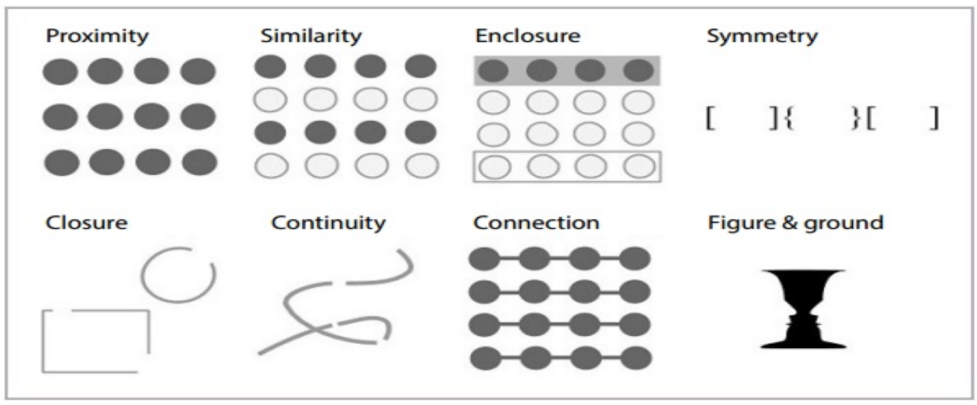


Figure 6.12: Gestalt's principles.

Here is what we notice from each of the illustrations [principles of Gestalt]:

- **Proximity**: we see three rows of dots instead of four columns of dots because they are closer horizontally than vertically.
- **Similarity**: we see similar-looking objects as part of the same group.

- **Enclosure**: we group the first four and last four dots as two rows instead of eight dots.
- **Symmetry**:we see three pairs of symmetrical brackets rather than six individual brackets.
- **Closure**: we automatically close the square and circle instead of seeing three disconnected paths.
- **Continuity**: we see one continuous path instead of three arbitrary ones.
- **Connection**: we group the connected dots as belonging to the same group.
- **Figure & ground**: we either notice the two faces, or the vase. Whichever we notice becomes the figure, and the other the ground

These principles allow us to perform many tasks such as reduce the noise from charts, choose the ideal aspect ratio, and show relationships between elements more clearly. Let's look at a dashboard, and see these principles in action.

How to select the right chart

The flowchart below can be helpful to understand the most suitable chart for your purpose.

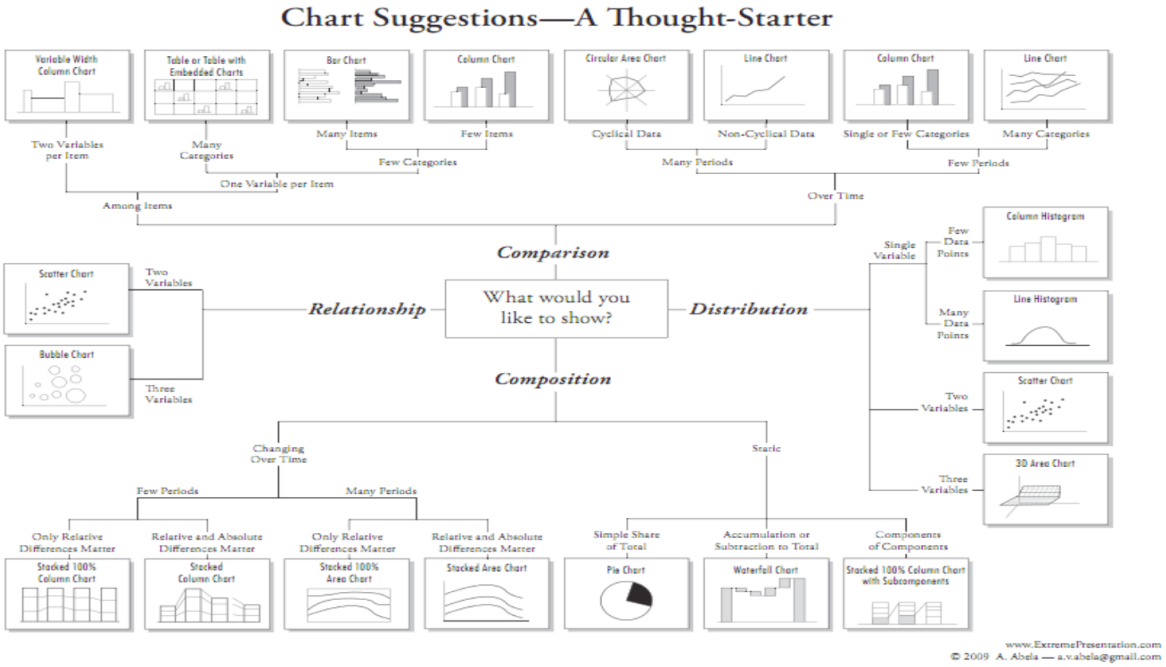


Figure 6.13: Make your choice.

Instead of using the chart, you can also find sources, like *Juice Analytics*, that let you choose interactively, means some filters, the best chart according to your necessities.

6.5 Interactive toolkits

- D3 is, without doubt, the most widely used interactive visualization framework, developed around 2011 by Jeff Heer, Mike Bostock and Vadim Ogievetsky. *Notes from the authors*: D3 is intentionally a low-level system. During the early design of D3, we even referred to it as a "visualization kernel" rather than a "toolkit" or "framework";
- Vega is a *visualization grammar* developed on top of `d3.js`. It specifies graphics in `JSON` format;
- Vincent is a Python-to-Vega translator;
- Bokeh is an independent Viz library focused more heavily on big data visualization. Has both `Python` and `Scala` bindings;

7 Supervised Learning

7.1 Introduction to Machine Learning

Machine Learning is an extended field characterized by many facets. The Arthur Samuel's definition can be considered the most meaningful one: "*Field of study that gives computers the ability to learn without being explicitly programmed.*". The aim of *Machine Learning* is to model into a computer the learning and adapting procedures that characterize the human way of thinking and processing information. Merely, the idea behind is to use computers to apply statistical and optimization algorithms to *automatically* identify pattern in data and/or classify them.
To get an idea of what *Machine Learning* does and consists of, take a look at a beautiful introduction to Machine Learning. In order to capture the essence of this subject, it speaks more than thousand words .
In the end, according to a reductive point of view, we may simply say that Machine Learning is born because we are **lazy** and we let the machine do the "work" for us.

Machine Learning is based on the definition of algorithms that allows the aforementioned procedures. Those algorithms can be distinguished either by the *learning style* (Supervised vs Unsupervised vs Semi-Supervised learning) or by their *similarity*.
In the next part of this chapter, we first give an introduction to *Supervised* and *Unsupervised* learning and then we go deeper into *Supervised Learning*. If you want to become a Machine Learning Master, you should have a look here.

7.1.1 Different aspects of Machine Learning

When we apply a *Machine Learning* method, one on the thing that we are mostly interested in is getting good predictions. This is not the only important aspect of the Machine Learning methods though.
The following list rough out some very important aspects for these methods:

- **Predictive accuracy**: we want our model to return correct result;
- **Speed and scalability**: the model should be efficient in order to be easily applied
  - Time to build the model
  - Time to use the model
  - In memory vs Disk processing
- **Robustness**: the method should not be too sensitive
  - Handling noise
  - Handling outliers
  - Handling missing values
- **Interpretability**
  - Understand the model and its decisions (*black box* vs white box)
  - Compactness of the model

7.1.2 Supervised vs Unsupervised Learning

In Table 7.1.2 we outline some differences/similarities between *Supervised* and *Unsupervised* learning.

The main difference to remark is that for *Supervised* learning, we use a *training* data with **known** labels and we test it on a*test* data **without** labels. For *Unsupervised* learning we do not have any labels. We are just trying to simplify/cluster the samples.

	Supervised	**Unsupervised**
Variables	Samples X and labels y	Samples X
Learning	Function $y = f(X)$ relate samples and labels. We would like to "learn" f and evaluate it on new data.	We want to compute a function $y = f(X)$ to give a *simpler* representation of the samples X.
Discrete labels	**Classification**	**Clustering**
Continuous labels	Regression	Matrix factorization, Kalman filtering, Unsupervised neural networks
Examples	• Is this image a cat, dog, car, house? • How would this user score that restaurant? • Is this email spam? • Is this blob a supernova?	• Cluster some hand-written digit data into 10 classes. • What are the top 20 topics in Twitter right now? • Find and cluster distinct accents of people in Lausanne
Techniques	• k Nearest Neighbors • Naïve Bayes • Linear + Logistic Regression • Support Vector Machines • Random Forests • Neural Networks	• Clustering • Topic Models • Hidden Markov Models

Table 1: Summary of the differences between Supervised and Unsupervised learning.

7.2 More details on Supervised Learning

7.2.1 Predicting from Samples

The **samples** are, most of the time, subsets of an infinite population. We are interested in a model that can **describe the whole population**, but since we only have access to a *sample* of it, it is not possible. Therefore we train on a training sample D and we denote the model found by $f_D(X)$, X being the features. Most of the datasets are **samples** from an infinite population, *i.e.* a subset of an **infinite** dataset. We would like to model the **whole population**, but only have access to a sample of it. So, we train on a training sample called D and we denote the model as $f_D(X)$ where X are the features and $y = f_D(X)$ the predictions.

7.2.2 Bias and Variance

The data-generated model $f_D(X)$ is a **statistical estimate** of the true function $f(X)$ (function working for the whole population). Therefore the model is subject to bias and variance.
The **Bias** is defined as the *expected difference* between the prediction of a model $f_D(X)$ and the true labels y:

$$\text{Bias} = \mathbb{E}\left[f_D(X) - y\right]$$

The **Variance** is defined as:

$$\text{Variance} = \mathbb{E}\left[\left(f_D(X) - \overline{f}(X)\right)^2\right]$$

where $\overline{f}(X) = \mathbb{E}\left[f_D(X)\right]$ being the average prediction on X.
Bias and Variance are very useful to understand if you are doing something wrong. Thus, it is important to understand what you are doing and not just applying "black-boxed" algorithms.

Trade-off between Bias and Variance

The tradeoff between bias and variance is due to model complexity, see Figure 7.1.

- **Complex models**: Many parameters, usually lower bias (the model is close to the model that generates the data) but higher variance (the predictions have high variability). It leads to the risk of overfitting. Merely, this kind of models describe well the sample population, likely they do not represent well the entire population though.
- **Simple models**: Few parameters, higher bias (far from the true model) but lower variance (less variability among predictions), It may ends up with the underfitting.

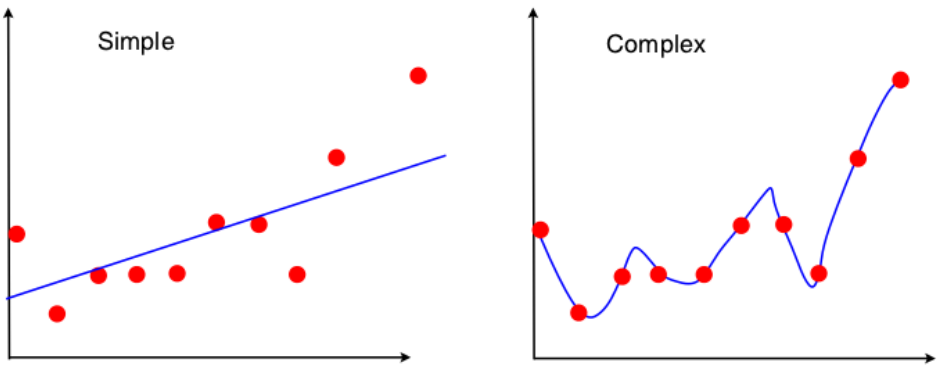


Figure 7.1: Illustrations of a simple model and a complex model on the same data.

For example, a linear model can only fit a straight line. A high degree polynomial can fit complex curves. Therefore this polynomial will work very well with the samples but not that well with the whole population. Thus a high variance is expected.

In order to take into account this trade-off, we introduce the total expected error is

$$\text{Bias}^2 + \text{Variance}$$

This error **balance** the contributions of the variance and the bias.

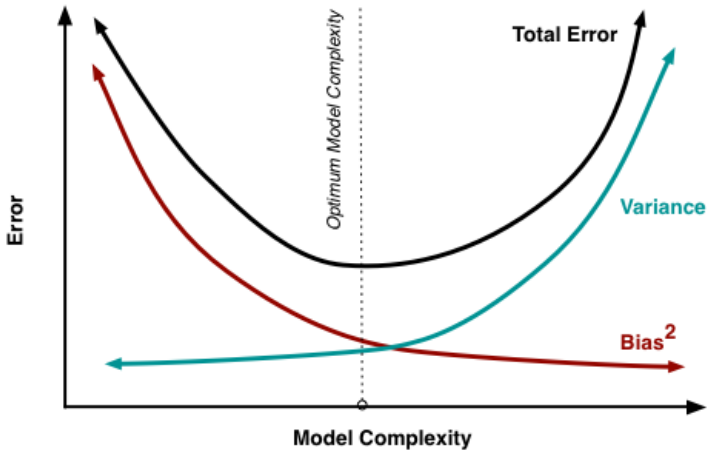


Figure 7.2: Illustration of the model complexity and the Bias Variance tradeoff. The optimum model complexity is when the total error is minimized.

When the Bias and the Variance are unbalanced, we use the terms:

- **overfitting** when the *Variance* strongly dominates. (Too much variation between models. Hence, the model does not work well on new data)

- **underfitting** when the *Bias* strongly dominates. (The models do not fitting the data well enough)

7.3 k-Nearest Neighbors

The kNN algorithm is a well known method used for classification and regression. Given a query item, the idea of the kNN algorithm is to find the k nearest neighbors (k closest matches) using a specific metric. Once we found them, we label the item such that it corresponds to the most frequent label in the neighbors. Figure 7.3 shows an example with cats and other animals.

Figure 7.3: Illustration of the kNN algorithm.

However, this very simple algorithm has one issue: the Data **is** the Model. This implies:

- No training needed.
- The accuracy generally improves with more data.
- Matching is simple and fairly fast if data fits in memory. (Can be run off disk)

Normally, the only parameter is k, the number of neighbors. But two other choices are important:

- Weighting of neighbors (e.g. inverse distance)
- Similarity metric.

7.3.1 kNN Flavors

The kNN algorithm can be used both for regression and classification. Table 7.3.1 gives the similarities/differences of the kNN algorithm for regression and classification.

Classification	**Regression**
The model is $y = f(X)$ where y is from a discrete set (labels).	The model is $y = f(X)$ where y is a real value.
Given X, we compute y being the majority vote of the k nearest neighbors.	Given X, we compute y being the average value of the k nearest neighbors.
We can also use a weighted vote of the neighbors.	We can also use a weighted average of the neighbors.

Table 2: kNN algorithm used for classification and regression: Differences and similarities. Usually, the weight function is the inverse distance.

7.3.2 kNN distance measures

For the kNN algorithm, we need to use a distance between the neighbors. The choice of the distance function can be very different depending on what you are looking for. We give here some examples:

Euclidean distance : Simple and fast to compute.

$$d(x, y) = ||x - y||$$

Cosine Distance : Good for documents, images, etc.

$$d(x, y) = 1 - \frac{x \cdot y}{||x||||y||}$$

Jaccard Distance : For set data

$$d(X, Y) = 1 - \frac{|X \cap Y|}{|X \cup Y|}$$

Hamming Distance : For string data

$$d(x, y) = \sum_{i=1}^{n} (x_i \neq y_i)$$

Manhattan Distance : Coordinate-wise distance

$$d)x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Edit Distance For strings, especially genetic data. See on Wikipedia for more information.

Mahalanobis Distance Normalized by the sample covariance matrix – unaffected by coordinate transformations.

$$d(x, y) = \sqrt{(x - y)^T S^{-1} (x - y)}$$

where S is the covariance matrix.

7.3.3 Choosing k

If we choose a **small k**, we will see a low bias but high variance. Firgure 7.4 shows whats happens with two different samples if we choose a small k.

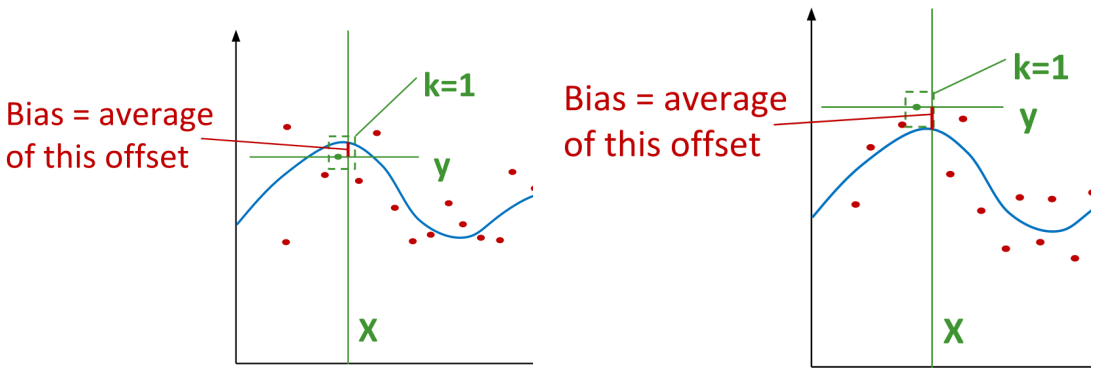


Figure 7.4: Test on two different samples of the kNN algorithm with k=1.

On the other hand if we choose a **large k**, we will see a high bias but low variance. Firgure 7.5 shows whats happens with two different samples if we choose a large k.

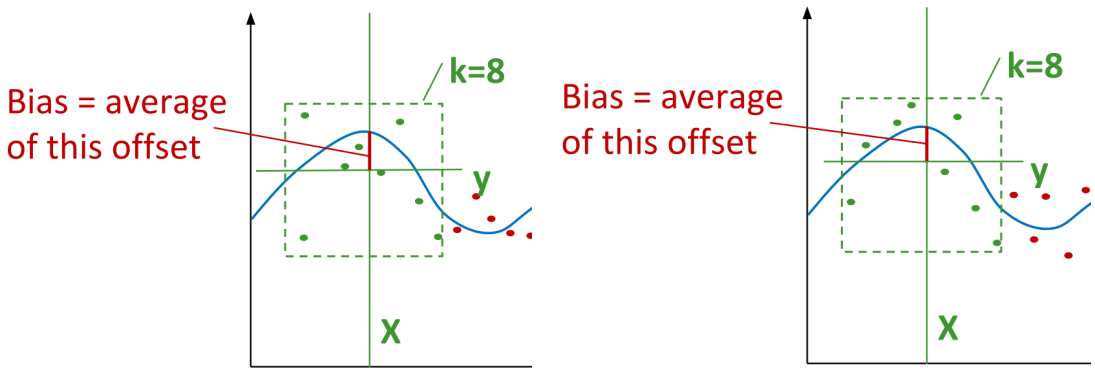


Figure 7.5: Test on two different samples of the kNN algorithm with k=8.

In practice

Use cross-validation! Break data into train, validation and test subsets. For example, you can choose a 60-20-20 % random split.

Predict For each point in the validation set, predict using the k-Nearest neighbors from the training set. Measure the error rate (classification) or the squared error (regression)

Tune Try different values of k, and use the one that gives minimum error on the validation set.

Evaluate test on the test set to measure performance.

7.3.4 kNN and the curse of dimensionality

The curse of dimensionality refers to phenomena that occur in high dimensions, from hundreds to millions, that do not occur in low-dimensional space. Data in high dimensions are much sparser than data in low dimensions. That means there are less points that are very close in the feature space. For example, the Euclidean distance scales as $\sqrt{N}$ with N being the dimension. Thus it is quite surprising that kNN works even in high dimensions. Luckily data are not random points in a high-dimensional cube. They live in **dense clusters** and near **much lower-dimensional surfaces**. Therefore, we can reduce the feature space in many different way to see the clusters appear.
Even if the Euclidean distance between two point is large they can be very *similar*. For example documents with the same few dominant words (with tf-idf weighting) are likely to be on the same topic.

7.4 Decision Tree

Decision Tree (DT) is a basic classifier acting like a **flow-chart having the following tree structure** :

- **Decision node**: specifies a test on a single attribute
- **Leaf node**: indicates the value of the target attribute
- **Edge**: split on one attribute
- **Path**: a disjunction of tests to make the final decision

It's constructed following a top-down approach in which, at each node, the data is split on one of their attribute. The prediction are obtained by following the "if-else" statement of each node and is given by the leaves , once the whole tree is traversed (Path).

Accuracy
Training accuracy : How many training instances can be correctly classified based on the available data? It is high when the tree is deep/large, or when there is less conflictual instances in the training instances. Never forget that higher training accuracy does not mean good generalization. Testing accuracy: Given a number of new instances, how many of them can we correctly classify. The only way to score a DT is by counting the number of right elements it predicts on the test set, since the prediction are not "weighted" by a probability of correctness.

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no

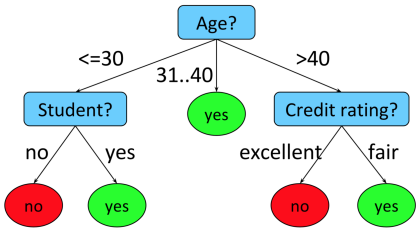


Figure 7.6: Dataset and the Decision Tree related to it.

Construction (Top-down, divide-and-conquer strategy.)

1. At the beginning, all the samples are attached to the root.
2. Recursively, the (sub)sets are partitioned according to their most discriminative attribute (see section 7.4.1 for the ways of selecting of the attributes).
3. Stop when:
   - A node only contains identically labelled samples, this node becomes a leaf .
   - No more attributes are left for splitting, assign the most dominant label to the leaf.
   - No more samples left.
   - A depth threshold has been defined and has been reached.

The DT will continue to add attributes to its decision process until none are left. As it increases its precision (and then its depth), it starts over fitting the model. A threshold can be defined to stop the construction process earlier and limit this effect.

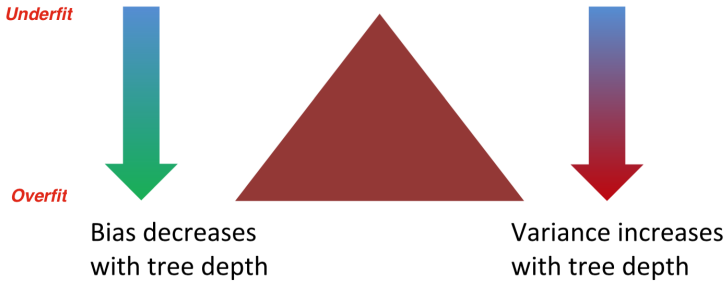


Figure 7.7: Bias and variance evolution with DT depth

Pros	**Cons**
+ A first ML approach ;)	- Sensitive to small perturbation
+ Can be enhanced in RF or BDT	- Retrained from scratch when new data are coming
+ Simple to understand and interpret	- Tend to overfit
+ Requires little data preparation	

7.4.1 Attribute selection

A big part of the DT model creation relies on choosing the good attribute on which to split the data in each node. One way to achieve this splitting relies on the concept of entropy, which describes the disorder a system and how impactful a feature can be.
For a set S with P positive predictions and N negative ones, its entropy is:

$$H(P,N) = -\frac{P}{P+N}\log_2\frac{P}{P+N} - \frac{N}{P+N}\log_2\frac{N}{P+N}$$

Note that:

- If P or $N = 0 \rightarrow H(P,N) = 0$, meaning **no uncertainty at all**
- If $P = N \rightarrow H(P,N) = 1$, meaning **maximum of uncertainty**

Entropy of the attribute A:

$$H(A) = \sum_i \frac{P_i + N_i}{P+N} * H(P_i, N_i)$$

Gain obtained by splitting the dataset S by attribute A.

$$Gain(A) = H(P,N) - H(A)$$

The gain indicates how a split on a certain attribute will influence our data set. The lower the entropy becomes, the greater is the gain, the more *organised* and *certain* become the data set.
The figure 7.8 illustrates how the entropy of an attribute is calculated.
In order to choose how to next split S, we compute the gain of each attribute A regarding S and choose the best.

Pruning The construction algorithm aforementioned doesn't filter out noise which may lead to over fitting. In order to circumvent this disagreement the Decision tree can be pruned.

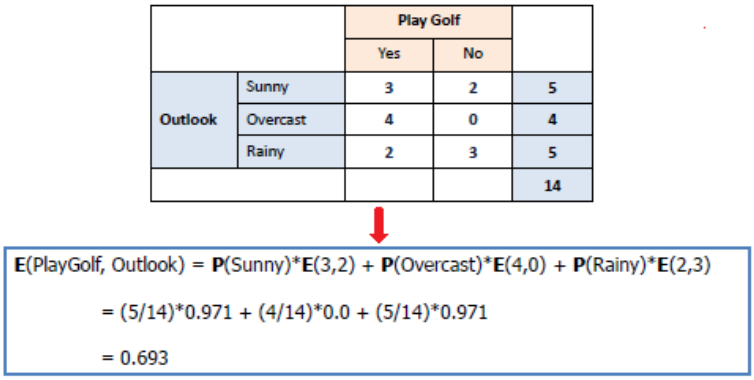


Figure 7.8: Example of entropy calculation.

7.4.2 Random Forest

A common problem when trying to build a model is to select significant features between all the available ones. We tend to think that *the more we have, the better it is*, even knowing the existence of the curse of dimensionality.
Random Forest exploits this previous idea and tries to automatize the feature selection process by randomly selecting subsets of features.
The main idea of Random Forest algorithm is to grow an arbitrary number of **Decision Trees**, each one based on a subset of m features, randomly chosen between the total p. It's an example of **weak learners** seen in the Ensemble method. These weak learners have a lower bias (because of lower number of feature).
Ensemble methods can be compared to crowd-sourced machine learning algorithms in the sense that we use a collection of weak learners and combine their results to make a better prediction. There are several different types of ensemble methods :

- **Bagging :** Train learners in parallel on different samples of the data and then combine the results by voting/averaging for discrete/continuous outputs.
- **Stacking :** A first series of learners is trained on the samples, then a combiner algorithm is trained to make a final prediction using the outputs of the first series of learners.
- **Boosting :** (see next section)

Pros	Cons
+ Popular	- Not state-of-the-art
+ Easy to implement	- Needs many passes over the data
+ Easy to parallelize	- Tend to overfit

7.4.3 Boosted Decision Trees

Variant of Random Forests. Instead of performing the training of all DTs independently on a weak subset, train learners on the filtered outputs of other learners. The building of

the ensemble is incremental where each new model instance is made to emphasize instances that previous models mis-classified. The accuracy yielded by boosting can be better than bagging but it also tends to overfit the training data. This is called **Boosting**.

- Low variance, because of the small trees handling small numbers of features
- Low bias, reduces by the boosting

Opposed to Random Forests that usually trains tens of medium-sized trees, Boosted Decision Trees works on smaller trees in greater numbers.
On the side of performances, even if they show good results, the big drawback of Boosted Decision Trees is their slow execution compared with the parallelized Random Forest.

7.4.4 About model transparency

A recurrent argument of using Decision Tree (DT) or Random Forest (RF) in industry is their transparency compared to state-of-the-art Deep-Neural-Network (DNN) that are often "black-boxed". This argument must be carefully taken. Even if by their very nature DT are more understandable by a human than DNN, the more features and Trees we had, the more complicated it becomes and the less transparency we have even for DTs. Figure **??** shows the size of a common industry implementation of RF and BT where there are no more possibilities of human understanding.

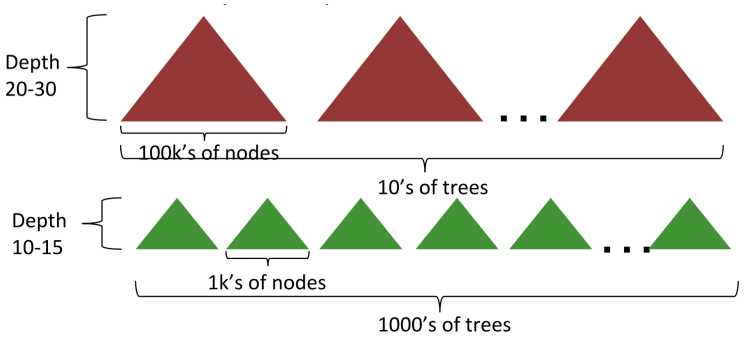


Figure 7.9: Standard size of RF and BDT implementations. In red: RF with big parallel DT. In green: BDT pipeline with lot of small DT

7.5 Linear Regression

Linear regression model produces a prediction equation:

$$\hat{y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j$$

or in matrix notation

$$\hat{y} = X\hat{\beta}$$

Where $X = \begin{pmatrix} X_{11} & \cdots & X_{1N} \\ \vdots & \ddots & \vdots \\ X_{M1} & \cdots & X_{MN} \end{pmatrix}$ is the input data, more precisely, rows of X are distinct observations and columns of X are input features.The $\hat{\beta}_j$ are the coefficients of the model.

7.5.0.1 Least Squares Solution

The most common measure of fit between the line and the data is the least-squares fit because if we start from the hypothesis that the points are the addition of a line with Gaussian noise, the least squares corresponds to the maximum likelihood solution.

7.5.0.2 Overfitting

A common mistake that leads to overfitting is to ignore feature selection, in fact having more features doesn't mean getting better results. Therefore it's always a good idea to carefully select features to improve model accuracy. More advanced models of regression include forms of feature control such as ridge regression or Lasso regression that include embedded regularizers.

7.5.1 Statistic validation

As a linear model can be evaluated for every possible existing dataset, it is not enough to compute it, but the existing of a meaningful linear relationship in the data must be determined beforehand.

7.5.1.1 R^2-value

R^2-value describes how much of the total variance is reduced when we include the line as an offset. It computes the distances between the real y and the predicted $\hat{y}$, and compares them with the distance between real y and the mean $\bar{y}$. It's therefore a good indicator on how a line fits our data.

- if $R^2 = 0$, then there is no difference between the linear model and the trivial mean $\bar{y}$. Then we conclude that there is not any evidence of linear relationship in our dataset and **we cannot use the model**.
- if $R^2 = 1$, then the data perfectly aligned on the linear regression line and **the model is perfect**.

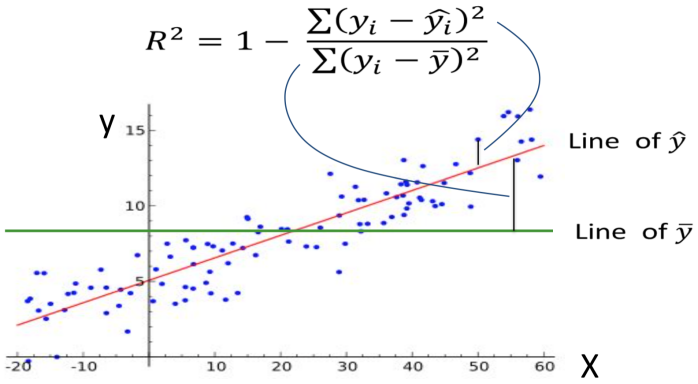


Figure 7.10: Graphical meaning of R^2-value.

7.5.1.2 p-value

From the Distribution of Fisher (F-Distribution) we can derive a p-value which is, as usual, the probability that the observed data validates the null hypothesis which is, in our case, the hypothesis *that there is no linear relationship in the data.* For $p < 0.05$ we conclude, as usual, that it is very unlikely that the data were produced by the null hypothesis and we then accept the hypothesis *the data follows a linear relationship.*

8 Applied Machine Learning

In this chapter, the pipeline of a *Machine Learning* process is explained [4]. Before discovering in-depth the ways which *Machine Learning* can be applied, we summarize briefly, in Figure 8.1, the time usage of the *ML* pipeline according to different approaches.

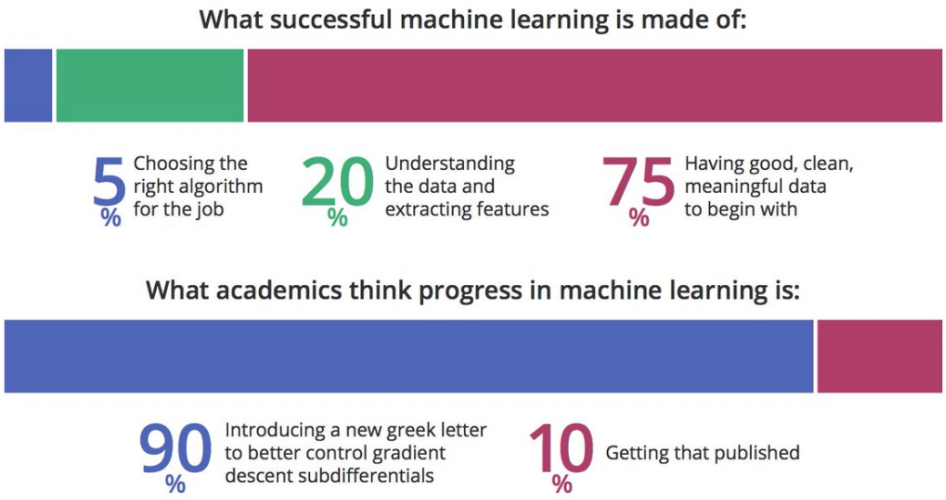


Figure 8.1: Machine learning *time usage*.

The showed pipeline refers to the *Classification* problem, it can be generalized for regression though. The Figure 8.2, clearly describes the main steps of the process.

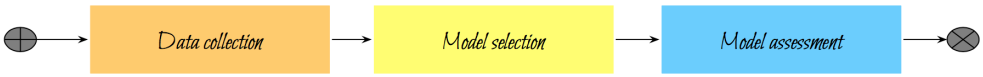


Figure 8.2: Classification *pipeline*.

8.1 Data Collection

The first step is the collection of data for a *Classification* task. In particular, it consists in defining the *attributes* that describe the data item of interest and the class label. In order to do that, a deep knowledge of the data domain is required. Furthermore, one of the most challenging aspects of the classification's *data collection* is to get the class label for the items. In fact, most of the time the labels are missing. Trying to avoid and reduce this problem, many solutions have been proposed and most of them involve the interaction with humans. At this point be creative can be an extra-point, in fact you can use your imagination to try to figure out systems that allowed you to get the data.

In Figure 8.3, we see more in-depth the pipeline of the *Data Collection* step.

[4] A few interesting things can be read on GitHub repository of Paper-we-Love

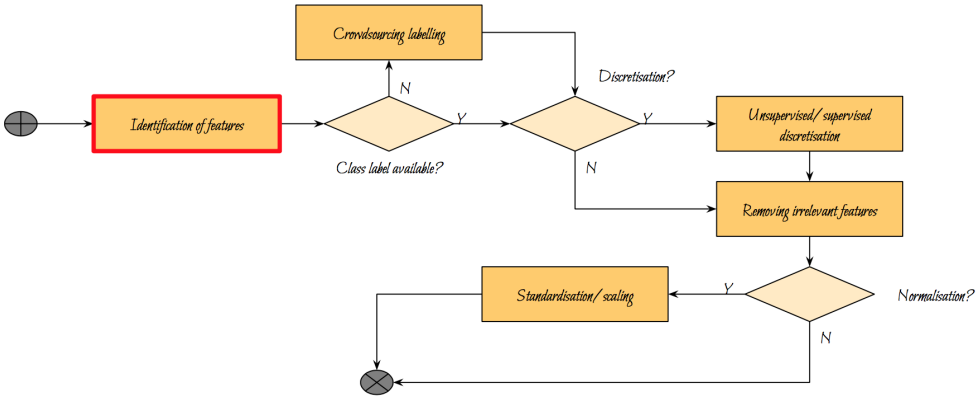


Figure 8.3: Data collection *pipeline*.

8.1.1 Identification of features

Since the feature describes the item we need to classify, their identification is a fundamental step in our pipeline. In particular we can distinguish between different types of features:

- *Numerical* (e.g., age, temperature ...)
- *Ordinal* (e.g., phone code ...)
- *Categorical* (e.g., student, weather ...)

Moreover, new features can be generated from simple *stats*, this process is known as *Feature engineering* and is considered a form of *art*. It means that there are not specific rules to follow, therefore the best suggestion is to look for people who have already attacked the same problem so that you can come up with new useful and powerful ideas.
Some classifiers, often, require categorical features. It implies that that the features should undergo a *pre-processing* known as *discretisation*. This procedure could be divided into two:

- **Unsupervised**: does not take class information into account
  - *Equal width*: divides the range into a predefined number of bins. The obvious weakness of the equal-width method is that in cases where the outcome observations are not distributed evenly, a large amount of important information can be lost after the discretisation process.
  - *Equal frequency*: divides the range into a predefined number of bins so that every interval contains the same number of values. For equal-frequency, many occurrences of a continuous value could cause the occurrences to be assigned into different bins. One improvement can be after continuous values are assigned into bins, boundaries of every pair of neighbouring bins are adjusted so that duplicate values should belong to one bin only.
  - *Clustering*: The advantage of using clustering as discretisation is that it can be performed on all the features at the same time, capturing in this way possible interdependencies of the features. Sometimes discretisation can even improve the performance of algorithms that, theoretically, do not need it.

- **Supervised**: takes class information into account, measuring the dependency of a discrete interval of values w.r.t. the class label as described below
  - Test the hypothesis that two adjacent intervals of a feature are independent of the class
  - If they are independent, they should be merged
  - Otherwise they should remain separate
  - Independence test: χ^2 statistics

The feature *pre-processing* and selection have changed over the year. Before 2012 (but still very common today) a clever design of the feature was considered the optimal recipe to make the model successful. After 2012 [5], the features and the model are learned together, in what is called mutually reinforcing.

8.1.2 Data labeling

Collecting lots of data is easy whereas labeling data is time consuming, difficult and sometimes even impossible. It is considered an extreme hard problem, moreover, based on subjective parameters. Nonetheless, there are ways to go beyond the limits of labeling, like the *crowdsourcing*. The latter is, generally, represented by a platform where data is passed and the human label the interested item. There are many platforms for doing it (*CrowdFlower*, *ClickWorker*, *MechanicalTurk*). Here, Figure 8.4, an example, that shows the standard procedure of these platforms, follows . Consider we want to know whether a web page is or not reliable.

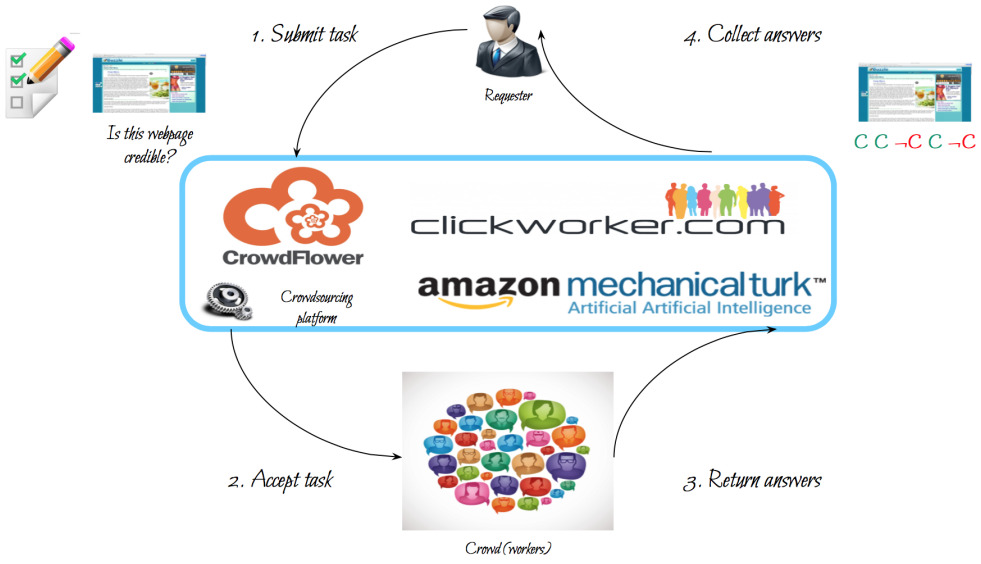


Figure 8.4: *Crowdsourcing* process.

You, the *requester*, pass the data to the platform, once the task has been accepted the *Crowd* (workers) return the answer. Before explaining how the answers are aggregated to finally label the item, it is important to consider the vast kinds of workers than can be faced.

[5] Krizhevsky's publication on ImageNet classification is considered as a turning point: Krizhevsky: ImageNet Classification with Deep Convolutional Neural Networks

Truthful	**Untruthful**
+ *Expert*	- *Sloppy*: gives many wrong answers due to their knowledge limitations or misunderstanding
+ *Normal*	- *Uniform spammer*: gives random answers to any question - *Random spammer*: keeps the same answer to every question

The Figure 8.5, shows how each type of worker tends to contribute to the *true positive* and *true negative* rates.

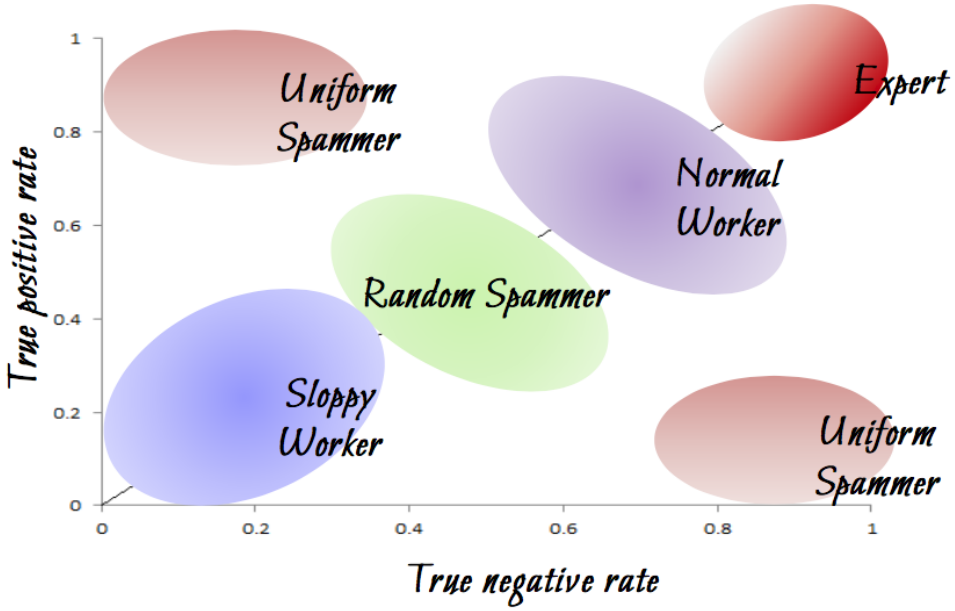


Figure 8.5: *Workers* outputs quality.

When the *crowd* has labeled the item, so all the answers are collected, we proceed aggregating them. In particular, the label assigned to the item is the one that registers the highest number of occurrences, Figure 8.6.

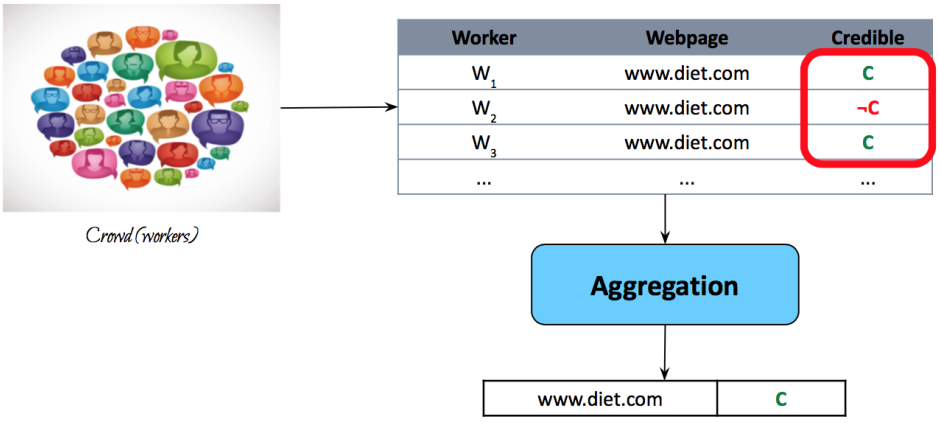


Figure 8.6: Answer *aggregation*.

8.1.3 Feature Selection

The aim of the *Feature Selection* procedure is to reduce the number of N features to a subset with the best $M < N$. The number of all the possible combination is 2^N, when N gets larger the required time to validate all the possible subsets of features sharply increases. Hence, there are two different available solution:

- **Filtering**: ranks features according to their predictive power and select the best ones
  - \+ Independent of the classifier (performed only once)
  - – Independent of the classifier (ignore interaction with the classifier)
  - – Assume features are independent

  Defining X as a feature and Y as the class label. The ranking of the features is determined in different ways according to the kind of attribute:

  - **Numerical**:
    - *Pearson Correlation Coefficient*

$$\rho = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

      Remark: This metric only captures the linear relations! Furthermore, defining the correlation threshold is subjective (which is a good correlation value? 0.7 is enough?).
    - *Mutual information*: measures the information that X and Y share. In particular, it measures how much knowing one of these variables reduces uncertainty about the other.

$$I(X,Y) = H(Y) - H(Y|X) = H(X) + H(Y) + H(X,Y)$$

$$H(X) = -\sum_i P(x_i)\log_2 P(x_i)$$

$$H(X,Y) = -\sum_i \sum_j P(x_i, y_j) \log_2 P(x_i, y_j)$$

In particular,

$$\begin{cases} If \quad \text{I(X,Y)} = 0 & X \quad does \quad not \quad tell \quad anything \quad about \quad Y \\ Elif \quad \text{I(X,Y)} = \max & X \quad tells \quad everything \quad about \quad Y \end{cases}$$

- **Categorical**:
  - χ^2 method: Different to correlation, the chi-square test checks the independence of the class and the feature, without indicating the strength or direction of any existing relationship. It is very powerful

  A very important thing to keep in mind is that **collectively relevant features may look individually irrelevant!**. Hence, it is important to try to figure it out.

- **Wrapper**: iteratively **adds** features, using cross-validation to guide feature inclusion and stopping when there is no improvement
  - \+ Interact with the classifier
  - \+ No independence assumption
  - − Computationally intensive
- **Ablation**: iteratively **remove** features, using *cross-validation* to guide feature inclusion and stopping when there is no improvement
  - \+ Interact with the classifier
  - \+ No independence assumption
  - − Computationally intensive

Remark: Beware of trusting correlations in a blind way!! *Correlation* is not *causation*.

8.1.4 Feature normalization

Some classifiers do not manage well features with very different scales. Features with large values dominate the others, and most of the classifiers tend to capture and optimize the attribute that vary the most.

- **Standardization**: assumes that the data has been generated by a Gaussian process

  $$X_i' = \frac{(X_i - \mu_i)}{\sigma_i}$$

  where X_i is a feature, μ_i its mean and σ_i its standard deviation. The new feature has $\mu_i = 0$ and $\sigma_i = 1$.

- **Scaling**: if the data has outliers, they scale the *normal* values to a very small interval

  $$X_i' = \frac{(X_i - m_i)}{(M_i - m_i)}$$

  where X_i is a feature, m_i its minimum and M_i its maximum. The scaled belongs to the interval [0,1].

Before choosing the kind of normalization ti apply, we should understand what data we are using.

8.2 Model selection

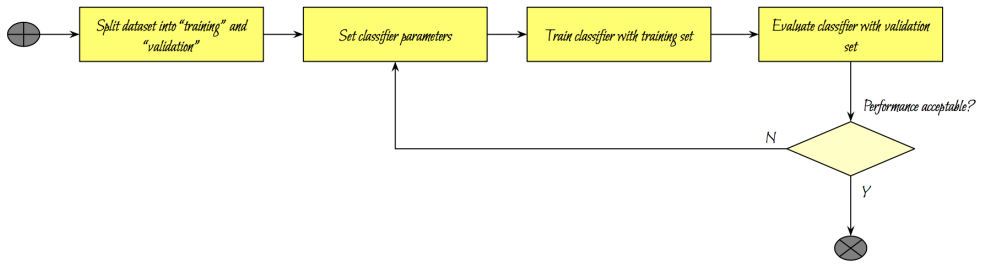


Figure 8.7: Model selection *pipeline*.

When we refer to the choice of a model, we do not only refer to the right classifier to pick but we also have to tune the parameters that characterize it. These parameters can be:

- *Regularisation factor*: how complicated the model can grow (i.e. Ridge regression)
- *Threshold*
- *Distance function* (i.e. KNN)
- *Number of neighbours* (i.e. KNN)

Moreover, it is necessary to define a metric that measures the performances of our model. We call the metric as *Loss function*, and we distinguish between:

- *Categorical output*
  - **0-1 loss function**: counts how many labels are predicted correctly

$$J = \sum_{i=1}^{N} \#(y_i \neq f(x_i))$$

    See section 8.2.1 for more advanced cathegorical loss function.

- *Real value output*:
  - **Mean Squared Error (MSE)**

$$J = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

  - **Mean Absolute Error (MAE)**

$$J = \frac{1}{n} \sum_{i=1}^{n} |y_i - f(x_i)|$$

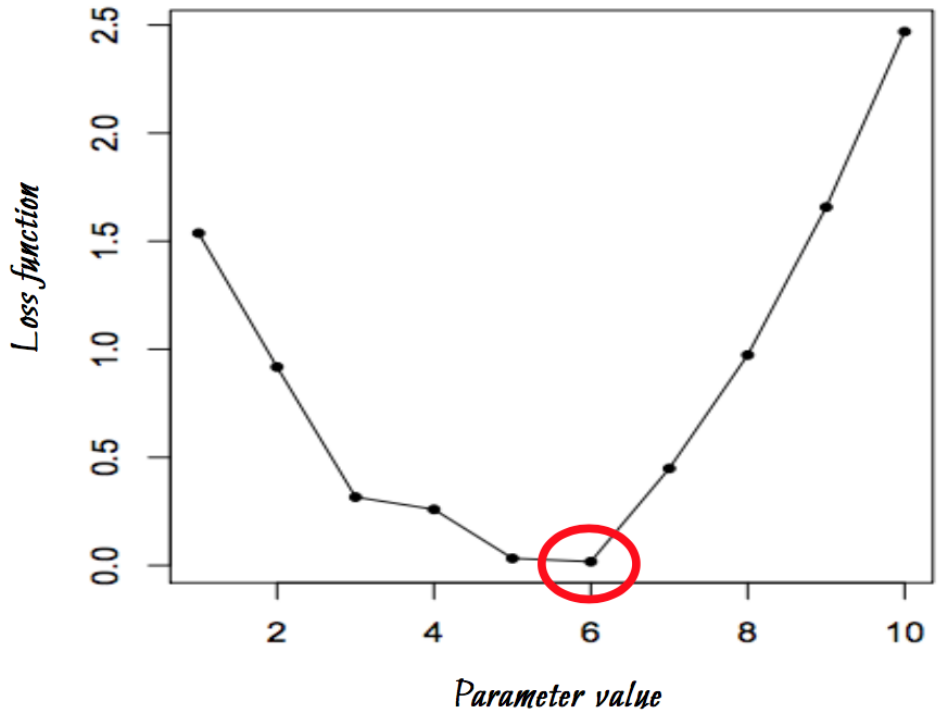


Figure 8.8: Loss function as function of the parameter.

There are many other kinds of loss functions, look up for them!

8.2.1 More performance metrics for the binary classification

For categorical binary classification, the usual metrics consider four types of errors, Figure 8.9:

- *True Positive*: positive examples classified as positive
- *True Negative*: negative examples classified as negative
- *False Positive*: negative examples classified as positive
- *False Negative*: positive examples classified as negative

		Class	
		A	B
Classified	A	TP	FP
	B	FN	TN

Figure 8.9: Confusion matrix.

According to what we want to control with our model we want to choose the most appropriate metric:

- **Accuracy**: generally used when the classes are not skewed and the errors have the same importance

$$A = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{N}$$

- **Precision**:

$$P = \frac{TP}{TP + FP}$$

  High precision means that few irrelevant examples treated as important

- **Recall**:

$$R = \frac{TP}{TP + FN}$$

  High recall means that that the classifier is good at getting the positive class right

Let's see an example, Figure 8.10:

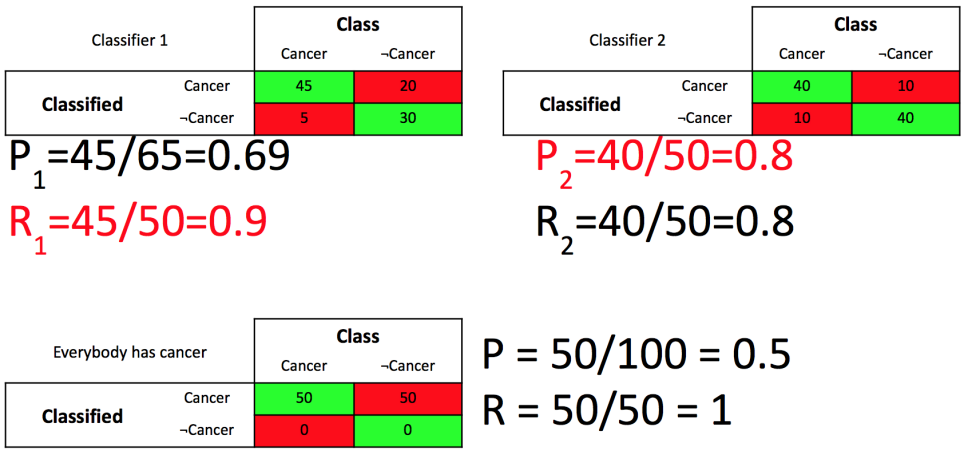


Figure 8.10: Example.

We see that the classifier 2 is more precise at hitting the cancer, in fact few patients erroneously diagnosed with cancer. Classifier 1 has a better recall, few patients with cancer are not diagnosed correctly. The classifier "Everybody Has Cancer" detects all the patients with cancer, but also causes that 50% of the patients with no cancer go home worried about their health status.
Due to the fact that to compare different classifiers we need a unique metric, we introduce

- **F-Score**:

$$F = 2 \cdot \frac{w_p P \cdot w_r R}{P + R}$$

That is the harmonic average of the *Precision* and the *Recall*. Whether one is more important than the other they can be differently weighted by setting w_p and w_r.
For a graphical vision we introduce the ROC *Area Under the Curve*, a single number that captures the overall quality of the classifier. It should be between 0.5 (random classifier) and 1.0 (perfect classifier).

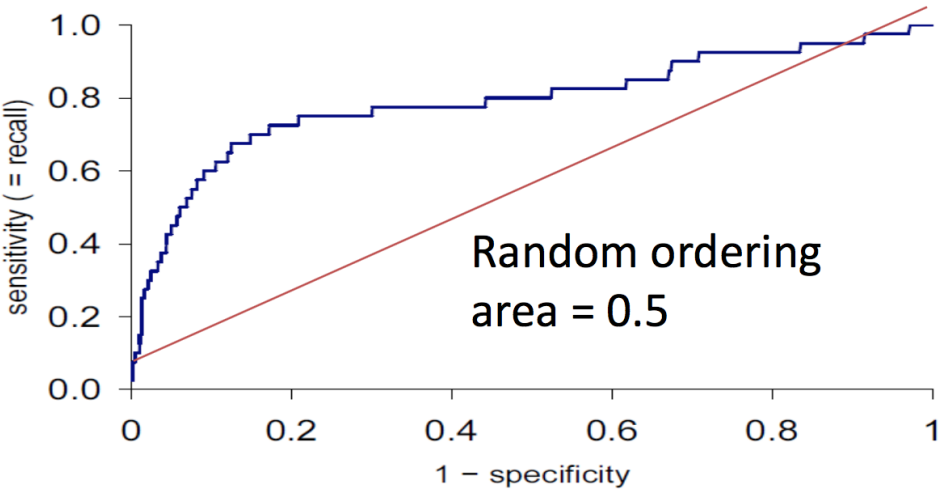


Figure 8.11: ROC AUC. *Y-axis*: true positive rate $= \frac{TP}{(TP+FN)}$, same as recall, *X-axis*: false positive rate $= \frac{FP}{(FP+TN)} = 1$ - specificity.

Since we have presented some usable metrics to verify the power of the model, we proceed to describe the pipeline used for the model selection.

8.2.2 Split data into Training and test set

Out inter set of data is split into two parts, generally, 60% training set and 40% test set. The former is used to learn the model, once it has been found it is evaluated, according to the chosen metric, on the test set. Take a look at Figure 8.12.

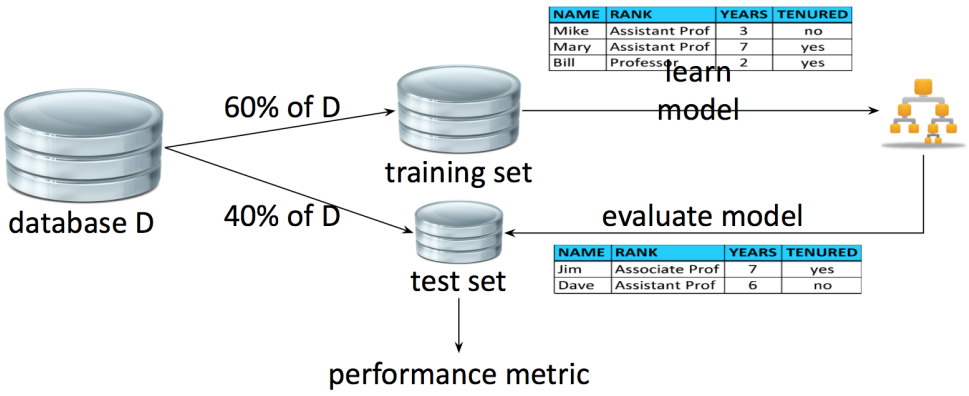


Figure 8.12: *Train* and *test*.

8.2.3 Tune parameters through the Cross-Validation and Bias Variance trade-off

We present two different types of *Cross-Validation*:

- *Leave-One-Out Cross-Validation (LOO)*: which train the model on all the data except one sample which the model is tested on. This operation is repeated for each sample, Figure 8.13. This form of *CV* is almost unbiased (=not optimistic) estimate of the true accuracy, however, is time consuming because it performs a number of

iterations equal to the number of samples in the data set. The evaluation of the model is the average of the errors obtained after each iteration.

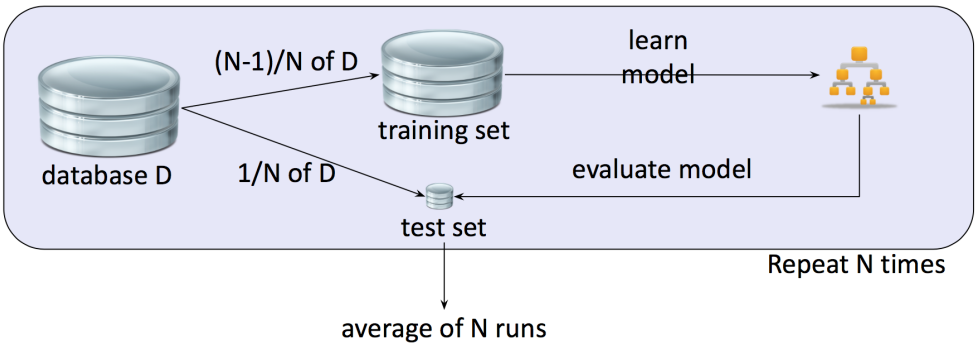


Figure 8.13: *LOO* cross validation.

- *K-Fold Cross-Validation*: which splits the data into K folds of equal size, Figure 8.14. In turn, $K-1$ folders are used as train set and the remaining as test. Also in this case, the evaluation of the model is defined as the average of the errors obtained after all the iterations. This kind of *CV* is a good compromise between having an unbiased estimate of the true accuracy and computation time. The most widely used K number is 10.

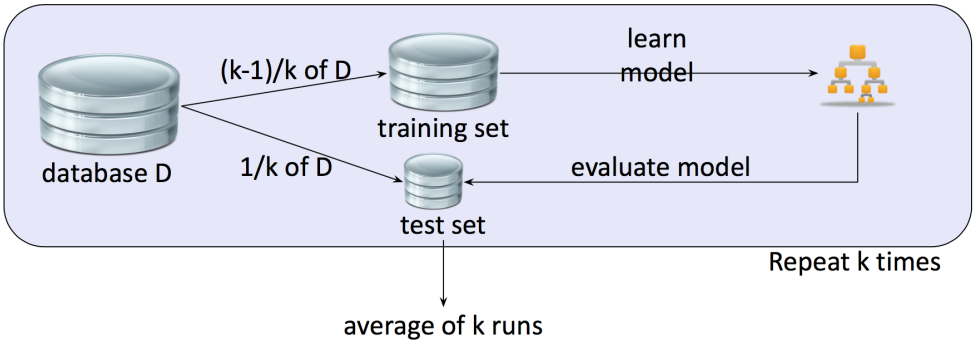


Figure 8.14: *K-fold* cross validation.

The bias and the variance of the model are other to important factors to take into account to select the model. In particular, they can be assessed by comparing the error metric on the training set and the test set, hence plotting the learning curves. In the ideal case, we want low bias (small training error) and low variance (small test error).
The other cases:

- *High bias - High variance*: is the worst case and having more data does not help
- *High variance - Low Bias* it turns out in overfitting, the training error is low, the test error high though. We can opt for reducing the complexity of the model, adding regularization parameter and more data may help
- *High bias - Low Variance*: in this case we have a high-training error and high test error, we fall in underfitting. A solution could be increasing the complexity of the model (adding more features), reducing the regularization parameter.

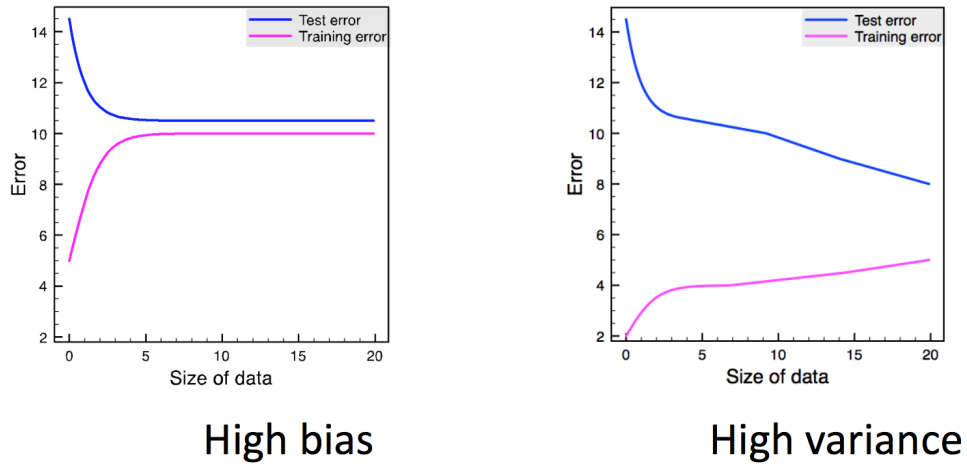


Figure 8.15: *Bias-Variance*: when bias is high (*underfitting*), more data does not help. When variance is high (*overfitting*), more data may help. .

8.3 Model assessment

Model assessment is the goal of estimating the classification accuracy of a fixed model (i.e., the best model found during model selection).
At this point, to evaluate the model we use all the data set and we repeat the following steps:

1. *Training and test*
2. *Leave-one-out* or K-fold cross-validation

This is a different goal than model selection. The performance obtained during model selection is a biased estimate of the true performance.

9 Unsupervised Learning

In this section we will look at how machines can learn from data without human intervention as opposed to *Supervised Learning*. The goal of Unsupervised Learning is to go from our dataset X to a function $f(X)$, a simpler representation of the data. We can already categorize Unsupervised Learning in two categories :

- Clustering for discrete data.
- Matrix factorization, Kalman filtering, unsupervised neural networks, etc... for continuous data.

9.1 Clustering in general

Going from a set of points with a distance metric between points to **groups** into some clusters. The goal is to perform same labelling for members that are close/similar to each other (Closeness can take on many shapes as we will see later), therefore members belonging to different clusters should be dissimilar.
The main use cases of clustering is having points in a high-dimensional space. Similarity/Distance is defined using some distance measures such as Euclidian, Cosine, Jaccard, edit distance, etc. The choice of such metrics is determinant regarding the performance and accuracy of the clustering, it is consequently a huge advantage to have a domain expert/specialist in order to choose the better metric.

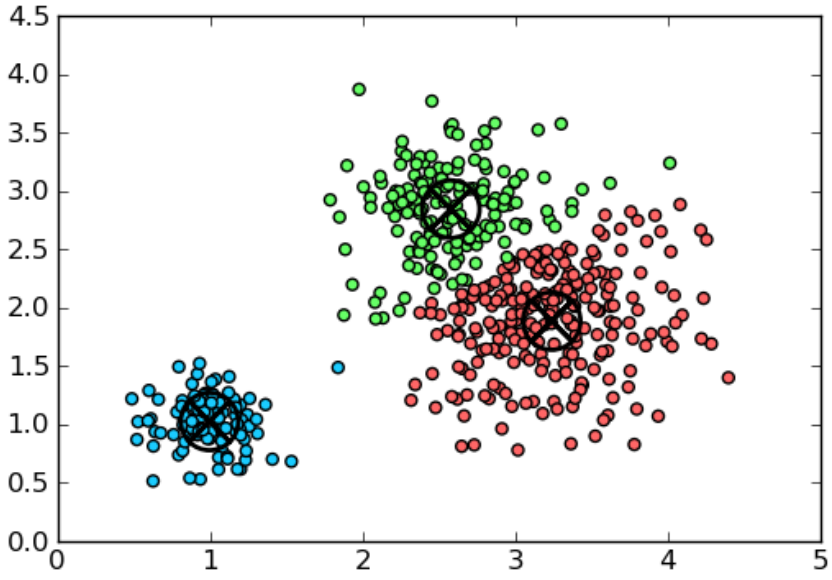


Figure 9.1: Example of clustering. (K-mean)

9.1.1 Terminology

There are several way to organize/apply clustering :

- **Point assignment**: Maintains a set of clusters. The points belong to "nearest" cluster.

  - **Hard clustering**: Every item is uniquely assigned to a cluster.
  - **Soft clustering**: Cluster membership is a real valued function, distributed across several clusters. This type of clustering is very agile and useful in many cases.

- **Hierarchical clustering**: Cluster coming from some sort of a hierarchy
  - **Agglomerative** (Bottom-Up): Initially, each point is a cluster (*singleton*). Then, we repeatedly combine the two "nearest" clusters into one.
  - **Divisive** (Top-Down): We start with only one cluster and recursively split it.
  - **Flat clustering:** As opposed to hierarchical clustering, there is no inter-cluster structure.

9.1.2 Characteristics of Clustering Methods

In order to better understanding clustering and when/how it should be applied, we have to take a peek at the characteristics of Clustering methods, namely :

- **Quantitative** : scalability (how many samples), dimensionality (how many features).
- **Qualitative** : Type of attributes (numerical, categorical, etc.), type of shapes (spheres, hyperplanes, etc.).
- **Robustness** : sensitivity to noise and outliers, sensitivity to the processing order.
- **User interaction** : incorporation of user constraints(e.g., number of clusters, max size of clusters), interpretability and usability i.e. how transparent is the model/ clusterization, black/white boxing c.f. trees/supervised learning.

These characteristics influence a great deal over clustering, and even by having them well defined clustering tasks are often found hard in easy looking situations.

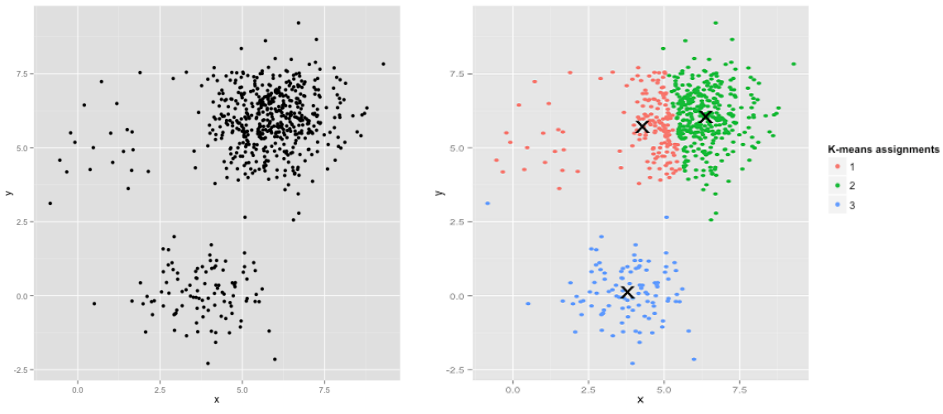


Figure 9.2: Image on the left is the raw data. We can see the three different clusters. The image on the left is the result of the K-means showing that outliers can be a problem for clustering the data.

Figure 9.2 shows what can happen with outliers and a simple clustering algorithm. Indeed, we see that on the left, the three clusters are quite visible. But the K-means algorithm

fails to find the three clusters. A very interesting discussion on the drawbacks on K-means can be found on StackExchange.

9.1.3 Examples of Clustering

9.1.3.1 Stereotypical Clustering

In high dimensionality clustering what may seem to be a right choice in high dimensionality may get bad when projected in 2D, where outliers often appear. Usually to obtain a good clusterisation, we have to accept that some points may not be classified well or even not classified at all because ultimately, the model and it's programmer cannot understand the process that generated the data. Figure 9.1 shows what is a typical stereotypical clustering.

9.1.3.2 Clustering for Segmentation

In the case of an image, it can be usef to segment/cut the image into multiple parts. For example, we would like to get the front, middle and background of an image. An example with K-means is given in Figure 9.3.

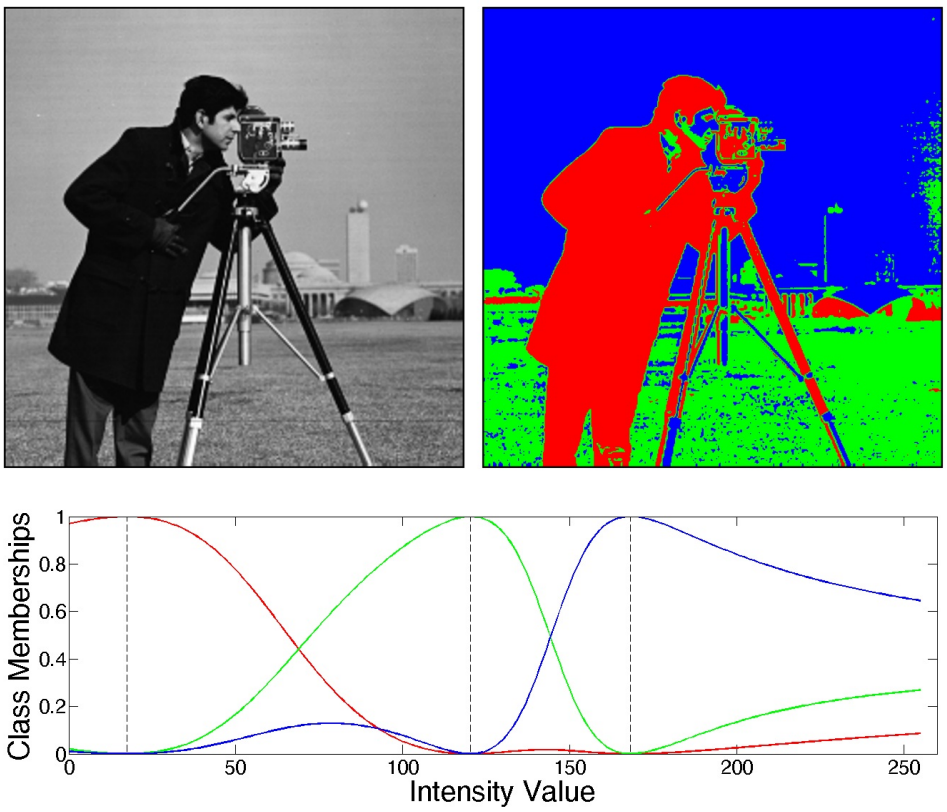


Figure 9.3: Example of segmentation on a black-and-white image. The algorithms used are K-means and Fuzzy C-means (or soft K-means).

9.1.3.3 Condensation/Compression

In Figure 9.4, we can infer several models/distributions that are contained into the data. But for a computer it is more difficult to see these underlying regression. Therefore we

can use Condensation/Compression to solve this problem. Condensation/Compression doesn't aim to cluster/model the underlying structure, but try to identify and subtract "submodels" in order to yield a simplification, a reduced variance "sample" of the data.

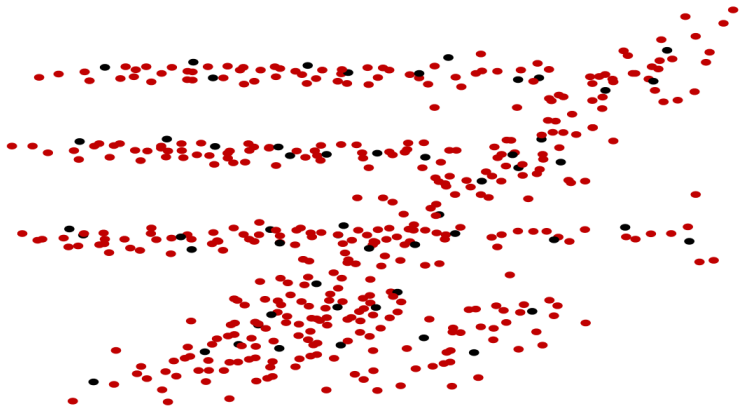


Figure 9.4: Example of data that can be compressed using clustering methods.

9.1.3.4 Read more

The Wikipedia page for clustering analysis gives different algorithms for clustering and explain different evaluation methods.

9.1.4 Cluster Bias

What is called Cluster Bias or Cluster hallucination comes from the human way of conceptualizing and categorize the world into categories (*exemplars*); we often tend to see cluster structures where there is not really. One of the most common example is the constellations. Nowadays, unsupervised learning is rather cheap. Thus, combined with cluster bias, instances of overuse or unjustified use of clustering are often encountered. A good data scientist has to ask himself if he has meaningful reasons to use clustering. People often assume that domain has discrete classes in it (ex. types of people). In reality the data is usually continuous.

9.2 Clustering : A hard problem!

Here our eternal curse of dimensionality strikes again. The data scientist is often tricked by how clustering looks easy when there are only two dimensions or when there is a small amount of data and it is in most cases true. Nevertheless most applications involve tremendously more than two dimensions and hight dimension spaces look very different from 2D spaces. For example, all points in high dimensions are about the same distance from each other.

9.2.1 Read more on this topic

Three articles on Alex Williams' blog, PhD student in Computational/Theoretical Neuroscience at Stanford University, are talking about clustering in different ways: What is clustering and why is it hard?, Is clustering mathematically impossible?, and Clustering is hard, except when it's not. These are really interesting articles on different aspects of

clustering. They are easy to understand, even the second with its mathematical approach of the clusters.

9.2.1.1 Music CDs

We can intuitively divide Music into categories. And we know that some customers will prefer a few categories. But **what are categories really**? We can for example represent a CD by a set of customer who bought it. Then similar (define similar) CDs will have a similar set of customers.
We see in this case that the representation is easy. But the dimensionality explodes quickly, *e.g.* Amazon. Therefore, choosing the right encoding is one of the determinant factor.

9.2.1.2 Documents

Encoding has a big impact on clustering, in this case the way we define a document as sets of words has a direct impact on the distance metric:

- Sets as vectors: cosine distance
- Sets as sets: Jaccard distance
- Sets as points: Euclidian distance

9.3 Hierarchical Clustering

The key operation of hierarchical clustering is how to repeatedly combine two nearest clusters. There are also several other questions that also arise when speaking about hierarchical clustering such as :

How is a cluster of more than one point represented ? The key problem is how the "location" of each cluster represented throughout the merging process in order to choose which pair of cluster is closest. In the **Euclidean case**, each cluster has a *centroid* computed from the average of its data points, note that a centroid doesn't have to be a data point it can be totally artificial. We can also use a *clustroid*, namely the existing data point "closest" to all of the other points according to the following metrics:

- Smallest maximum distance to other points.
- Smallest average distance to other points.
- Smallest sum of squares of distances to other points.

How is the "nearness" metric of cluster determined ? We can measure the nearness between clusters according to different approaches:

1. *Intercluster distance*: Minimum of the distances between any two points, one from each cluster.
2. *Cohesion* : Pick a metric e.g. maximum distance from the clustroid for example. And then merge clusters whose union is most cohesive. Cohesion can also be: *diameter* of the merged cluster (max distance between points in the cluster), *average distance* between points in the cluster, *density-based approach* (ex: for geolocation data) see after.

When should we stop combining clusters ? It is always difficult to decide when we should stop clustering. One way would be to compute some measure of *how well the clustering fits*. For example, we can use a sum of distances from cluster centres. Then, we can check if the graph of the error in function of the number of clusters has an elbow. Finally, we can choose the number of cluster minimizing this measure. Another way to check if the clustering is still good is to use the silhouette score.

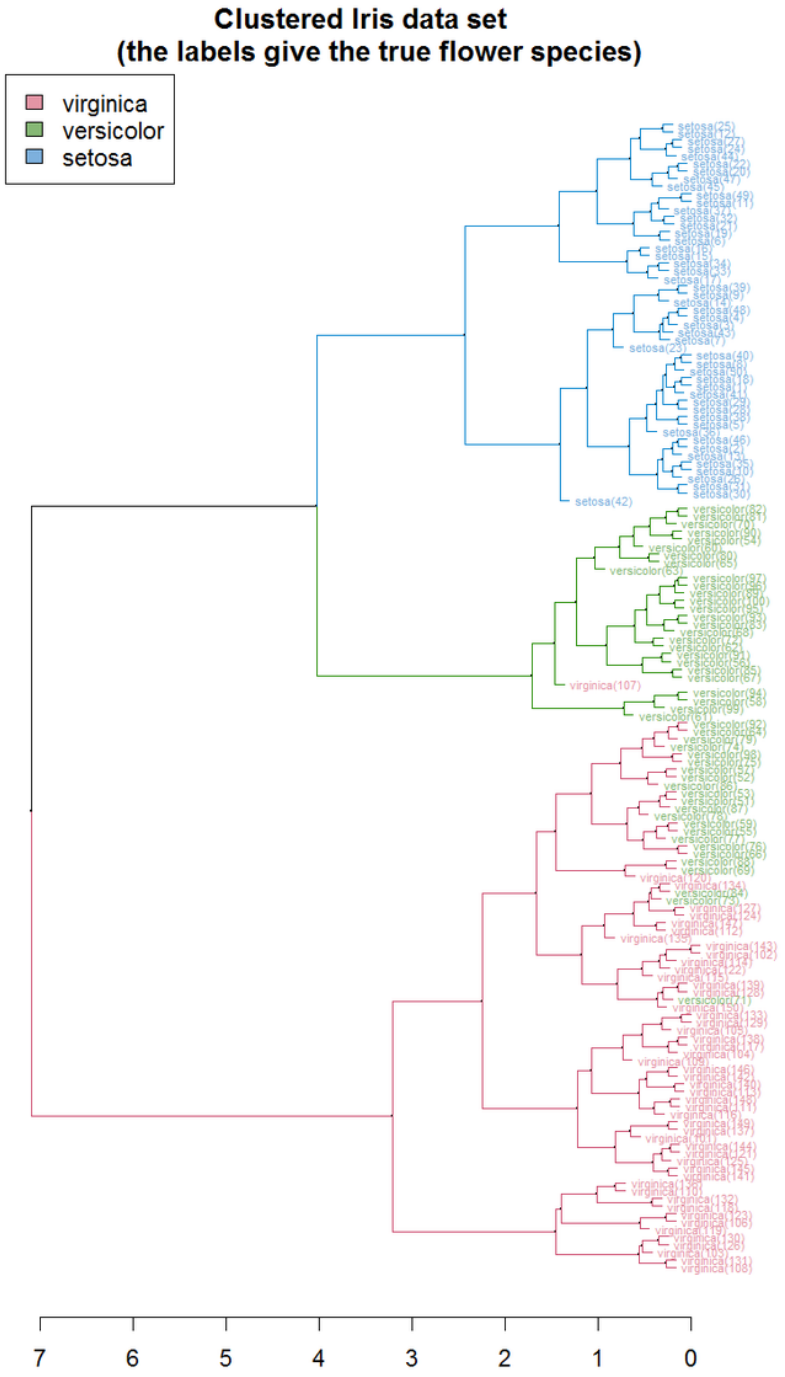


Figure 9.5: Dendogram of the Iris data set. The hierarchical clustering has been done using a **agglomerative** algorithm.

9.3.1 Implementation of hierarchical clustering

The naive implementation of the hierarchical clustering can be done using the pairwise distances between all pairs of clusters. But computing this pairwise distance and merging them has a total complexity of $O(N^3)$. A more clever approach using better indexing structures such as priority queues can improve performance up to a $O(N^2 \log(N))$ complexity. Nevertheless, it is still too complex for big dataset not fitting into the memory.

9.4 K-means Clustering

9.4.1 Standard K-means

The standard K-means algorithm is based on **Euclidean distance**, the quality measure is **intra cluster** only. It is a simple greedy algorithm that locally optimizes the quality measure. The algorithm resumes to the following two steps :

- **Finding the closest cluster center** for each item and assign it to that cluster.
- **Recompute the cluster centroid** as the mean of the items, having added the new item to the cluster.

The most challenging and impactful decisions when using K-means is to choose the number of clusters and also to pick the initial cluster centres. The latter can be done by sampling the input data or following more complex methods as described later. We also have to choose when to stop iterating by either :

- Defining a fixed number of iterations.
- Until no changes in assignments happen.
- Until there are only small changes in quality.

As we said previously, **initialization** is the most crucial part when running K-mean, we can choose to pick a random subset of k points from dataset or use methods like K-Means++ that iteratively constructs a random sample with good spacing across the dataset. However you choose the method for choosing your initial points, take into account the fact that optimal K-Means clustering is a NP-hard problem and that randomization helps avoiding bad configurations and saves up a lot of time/complexity.

9.4.2 K-means++

The idea of the K-means++ algorithm is the following:

Start Choose the first cluster center at random from the data points

Iterate
- For every remaining data point x, compute $D(x)$ the distance from x to the closest cluster center.
- Choose a remaining point x randomly with probability proportional to $D^2(x)$, and make it a new cluster center.

Intuitively, this finds a sample of widely-spaced points avoiding "collapsing" of the clustering into a few internal centers.

9.4.3 K-means properties

Here are some properties of the K-means algorithm:

- It's a greedy algorithm with random setup. Therefore, the **solution is not optimal** and varies significantly with different initial points.
- It has a very simple convergence proof.
- The **performance is in** $O(nk)$ **per iteration**, n being the total features in the dataset and k the number of clusters. It is quite good and it cannot be heuristically improved.
- There are many generalizations;
  - Fixed-size clusters
  - Simple generalization to m-best soft clustering
- As a "local" clustering method, it works well for data condensation/compression.

9.4.4 K-means drawbacks

Every time an algorithm has good properties, it will have drawbacks. Here are some of them:

- It often terminates at a **local optimum**.
- It needs a **distance function** to compute the mean.
- It needs to specify **K** in advance.
- It does not handle **noisy data and outliers**.
- Clusters **only** have **convex shapes**.

9.4.5 How to choose K? Elbow Method

In order to choose K, we can use the same techniques as when we have to stop the clustering in the hierarchical clustering. The idea would be:

- We execute the clustering for $K = 1, 2, 3, ...$
- We define J as the "average within cluster sum of squares"
- We plot J against K.
- We pick K such that $J(K) \simeq J(K + 1)$

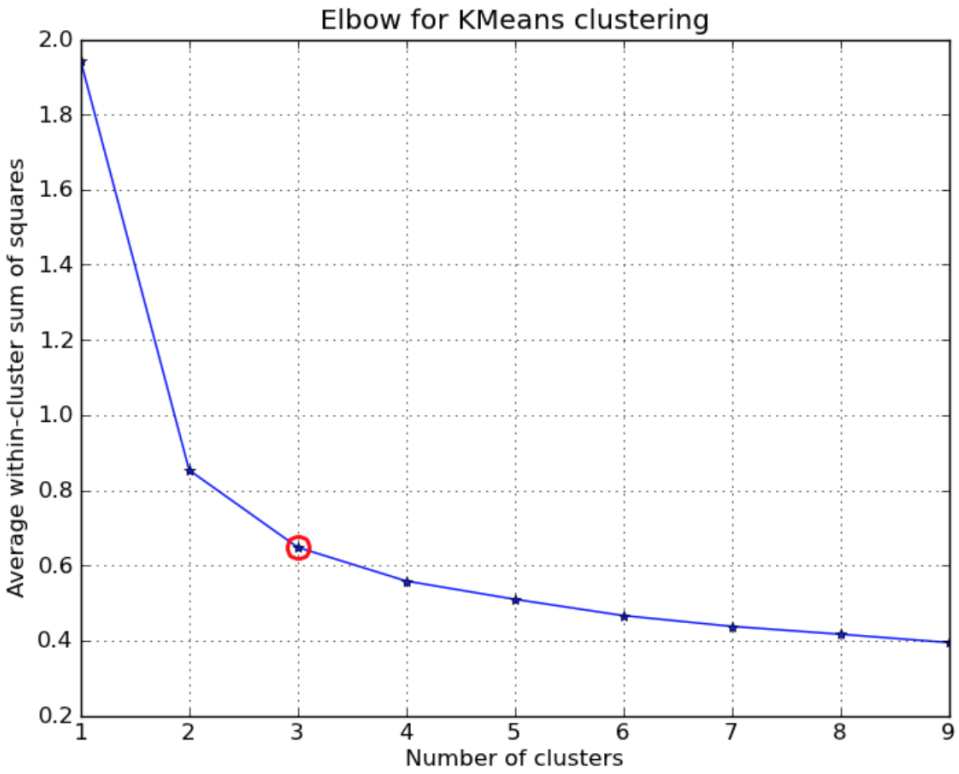


Figure 9.6: Plot of the within-cluster sum of squares. The chosen clusters are three. That's the point K s.t. $J(K) \simeq J(K+1)$

9.5 DBSCAN

The usual Centroid-based clustering methods, including k-Means, favor **spherical** clusters, but in the real world, the need to be able to cluster more intricate structures and complex shapes often arises. Therefore we need other clustering methods to avoid painful and difficult clustering with complex data such as the ones we can see below.

9.5.1 The algorithm

DBSCAN performs density-based clustering and concentrates on the shape of dense neighbourhoods of points. In order to define a neighbourhood density we need to introduce the notion of **Core points**. They are point that have at least *minPts* contained in a sphere of diameter ϵ around them. Core points are said to define *dense spheres* and can directly reach all neighbours in their $\epsilon - sphere$. All other points cannot directly reach other points. We also have to define several notions in order to understand DBSCAN properly:

- **Density reachability**: A point q is **density reachable** from point p if there exists a series of points $p_1, p_2, ..., p_n$ such that $p_1 = p$, $p_n = q$ and p_{i+1} is *directly reachable* from p_i. Meaning that all the points except p_n need to be *core points*.

- **Outliers** : all points that are not *density reachable* from any other. Be careful here as nothing is done to prevent the creation of clusters from a set of outliers, this outlier definition doesn't ensure safety and good clustering, even if the data is well

shaped. A good data analyst will look manually at the outliers in order to avoid creating clusters from outliers.

- **Density-connection** : We say that p and q are **density-connected** if there exists a point o such that both p and q are *density-reachable* from o.

We can now define a **cluster** as a set of points that are **mutually density-connected**, note that if a point is *density-reachable* from a cluster point, it is also part of the cluster.

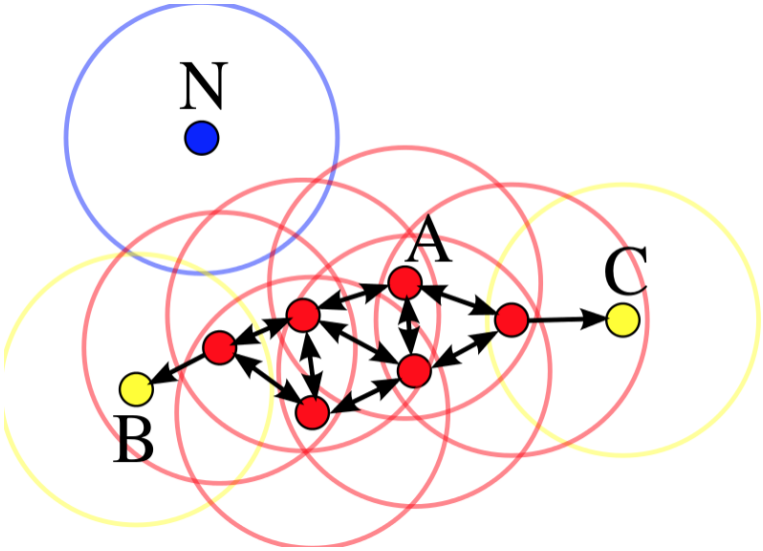


Figure 9.7: Points at A are core points. Points B and C are density-reachable from A and thus density-connected and belong to the same cluster. Point N is a noise point that is neither a core point nor density-reachable. (MinPts=3 or MinPts=4)

Here is a pseudocode version of the algorithm:

Algorithm 1: DBSCAN

Input : Dataset D, Sphere diameter eps, Minimum number of points for dense sphere MinPts.
Output: Clustering of the DataSet D.
begin
  C ← 0 ;
  for *each point* P *in* D **do**
    if P *is* visited **then**
      continue to next point ;
    end
    mark P as visited;
    NeighborPts = *regionQuery*(P, eps) ;
    if *sizeof(*NeighborPts*)* ≤ MinPts **then**
      mark P as noise;
    else
      C ← next cluster ;
      expandCluster(P, NeighborPts, C, eps, MinPts) ;
    end
  end
end

9.5.2 Performance

DBSCAN has to compute all pair point distances but it is using efficient indexing structures and assumes that the neighbourhood size is not too big in terms of points, therefore each neighbourhood query takes $O(n \log n)$ time. Ultimately the whole algorithm can be made to run in $O(n \log n)$ time. Nevertheless, note that fast neighbour search becomes progressively harder in higher dimensions and that fast implementations require that the data can be held in memory (therefore implying a $O(n^2)$ out-of-memory implementation). The silver lining here is that DBSCAN can catch different shapes for clustering, other than plain spheres.

9.6 Matrix Factorization

For problems with linear relationships between a response Y and feature data X we use the linear regression formula:

$$\hat{Y} = \hat{\beta}_0 + \sum_{i=1}^{p} X_j \hat{\beta}_j$$

Another appearance of the **"curse of dimensionality"** is that the quicker the dimension of Y grows, the quicker we *"run out of data"*, meaning that the model has more degrees of freedom than we have data. We encounter this kind of problem in recommendations or information retrieval for example, where features and responses are the same (e.g ratings of products, relevance of a term to a document) and where we have to deal simultaneously with thousands or millions of responses in coordination with thousands or millions of features. The way out of this kind of problem is that High-dimensional data will often have a simpler underlying structure in regard with the interests of the data analyst, for example :

- **Information retrieval:** Topics in documents, Themes in discussions, "Voices", styles in documents.
- **Recommendations:** User preferences, personality and demographics. General and principal product characteristics (reliability, cost ...).

All of the aforementioned can be approximated using a lower-dimensional linear model. The idea is that a subset of features can be used to predict the values of the others, practically **d** coordinates defines a unique point on a **d** dimensional plane and from these measurements, we can determinate the other coordinates. Such methods are used in systems such as User product preferences, CTR prediction, search engine optimization. Another aspect of the problem is that data may vary more depending on dimensions. Therefore, fitting a linear model to the top k *varying* or *strongest* direction gives the best approximation of the data, in a least-squares sense. So we want to isolate the *best* features.

In an Algebraic point of view, we want to approximate a high-dimensional and typically sparse matrix S by a product of two dense matrices A and B with low *inner* dimension. In the figure example, columns of B represent the topics of the n^{th} sample and rows of A represent the distribution of topics in the m^{th} feature.

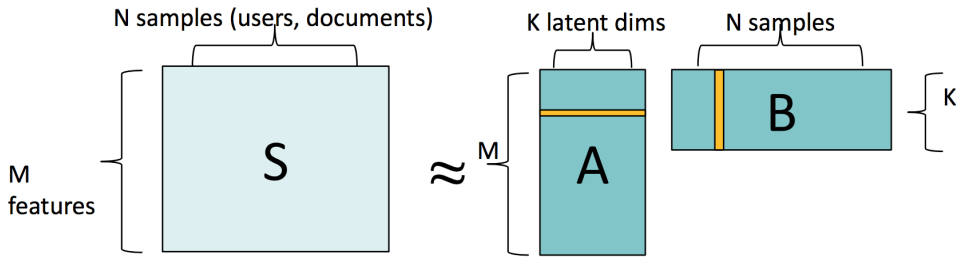


Figure 9.8: Matrix representation of the *matrix factorization*

For further readings and information about matrix factorization: Matrix Factorization: A Simple Tutorial and Implementation in Python or the article : Matrix Factorization Techniques for Recommender Systems.

9.7 Performance

Performance is needed for several reasons, namely :

- **Dataset size:** assuming model quality improves with size.
- **Model size:** bigger models generally perform better.

These above reasons are also true for power-law datasets.

9.7.1 Offline/Online Performance

There are two different dimensions of performance:

- **Offline: Model training:** Uses extensive amount of resources, we're typically talking about 1 to a few days of training time in order to yield the best possible model accuracy.
- **Online: Model prediction:** Uses a limited amount of resources (only memory and machines). The goal is usually to minimize latency and also online model size.

Usually *Model prediction* influences *Model Training* because models trained offline must fit and be fast in their deployment environment.

9.7.2 Algorithm improvements

In order to optimize performance one can algorithmically improve the construction of the models for example with *adaptive SGD methods*. This kind of improvement usually comes for free (it is only a matter of implementation) and allows to gains orders of magnitude time.

9.7.3 Computational improvements

A common way to improve running time in computer science is by optimizing the code (memory accesses, data structures, etc...) or leverage hardware (e.g using GPU's). Again you can better your performance by orders of magnitude. This kind of improvement is easily combined with algorithmic improvements. It's again a matter of implementation time and being able to make the right decisions in the scope of the problem to solve.

9.7.4 Cluster scale-up

Distribute model or data across machines over a network and parallelize computations. This requires knowledge of the algorithms and data in order to know which are the steps that can be distributed and to be able to reduce network load by several orders of magnitude.

10 Scaling Up

For the moment, we always made one big assumption: **All data fits on a single server**. Even more, all data fits in the memory. It's a realistic assumption for *prototyping*. But it becomes totally obsolete when moving to production. In this chapter, we will introduce the Big Data Problem as well as Spark.

10.1 Big Data Problem

Nowadays, Internet is the biggest source of data. And it is growing faster and faster. Even if computers are really powerful, Data Scientist encounters one problem: The Big Data Problem. Indeed, **Data is growing faster than computation speeds**.
We can see, for example, that CPU speed is stalling and that there is a bottleneck in storage. Therefore a single machine can no longer process or even store all the data. The only solution is to **ditribute** data over large clusters.

10.1.1 How much data do you need?

The answer depends on the question. But for many applications the answer is: **as much as we can get**. Big Data about people (text, web, social media, etc.) follow the power law statistics. Indeed, most of the features occur only once or twice. Therefore the distribution is a long tail distribution. And the long tail is really important. For example, Google knows that you're looking for something even if you checked on the web once or twice.
The number of features grows in proportion to the amount of data, *i.e.* doubling the dataset size roughly doubles the number of users we observe. Therefore, even one or two observations of a user improves predictions for them, so:

More data and bigger models => More revenue!

10.1.2 Hardware for Big Data

At the beginning, Google (and other Internet companies) used many low-end servers instead of expensive high-end servers. It is easier to add capacity and it is cheaper per CPU/disk. But there are problems:

- **Failures** (e.g. Google numbers)
  - 1-5% hard drives/year
  - 0.2% DIMMs/year (RAM)
- **Commodity Network** (1-20 Gb/s) speed vs RAM
  - Much more latency (100x – 100000x)
  - Lower throughput (100x-1000x)
- **Uneven Performance**
  - Inconsistent hardware skew
  - Variable network latency
  - External loads

These numbers are constantly changing thanks to new technology!

10.1.3 How do we split work across machines? (Map-Reduce)

How do you count the number of occurences of each word in a document?

We can use a **hash table**. It's an easy tool to use for these kind of task. But what happens if the document is really big? (For example, the web) We can simple **chunk** the document into multiple documents and create a hash table on each machines. At the end, we aggregate all the hash tables. **But** the machine that aggregates is the **bottleneck** in term of performance. And if this machine crashes, it's a disaster!

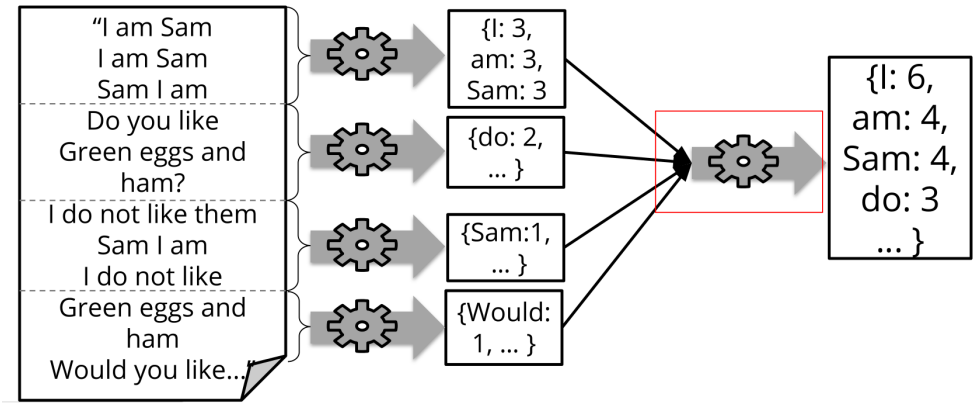


Figure 10.1: Usage of hash tables on multiple machines. We see that the bottleneck machine is the machine that aggregates the hash tables into one.

Therefore, we can use a more clever algorithm: **Divide-and-Conquer**. Each node is responsible os summing up a subset of the whole hash table.

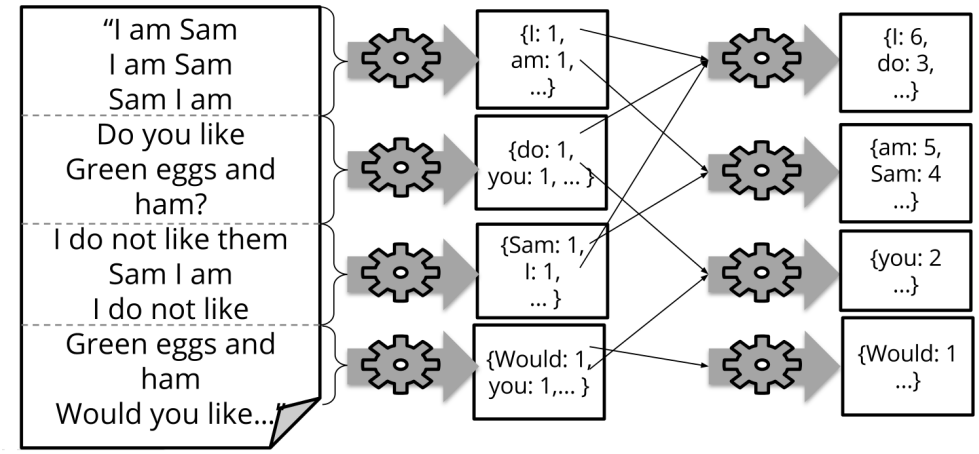


Figure 10.2: Usage of hash tables on multiple machines with Divide-and-Conquer technique. We don't have any bottleneck this time.

This technique is called **Map Reduce**. The first part is called map because we create a map for each part of the big document. Then we reduce it into a simpler hash table. This works well when using a lot of different nodes. In addition, each task is **idem-potent**. It means that the order of the word count doesn't matter. Thus, it is really easy to user without waiting the other nodes to finish.

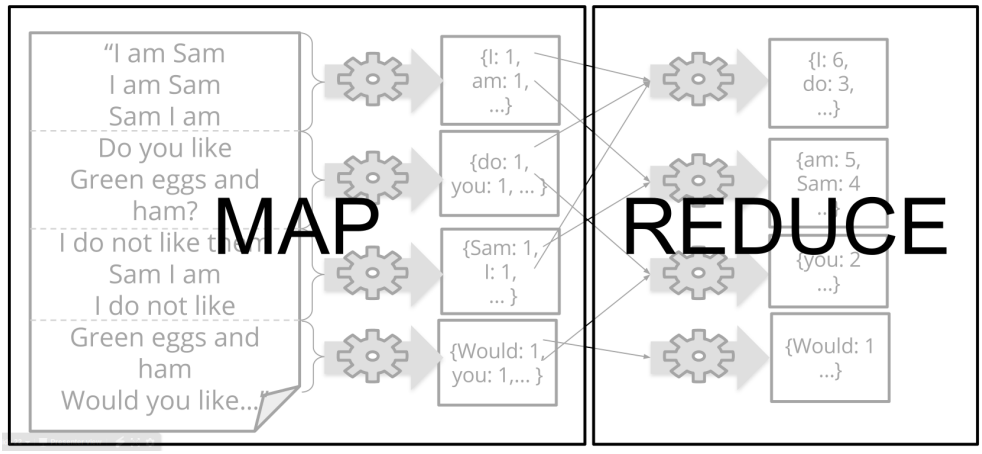


Figure 10.3: Map Reduce illustrated on the Divide-and-Conquer algorithm in Figure 10.2

10.1.3.1 What's hard about cluster computing?

How to divide work across machines?

- We Must consider network, data locality. (It's always better on its own machine.)
- Moving data may be **very** expensive!

How to deal with failures?

- 1 server fails every 3 years. Therefore, 10K nodes will give 10 faults/day.
- Even worse: stragglers. The node has not failed but is slow.

How to deal with failures? **Just launch another task!**

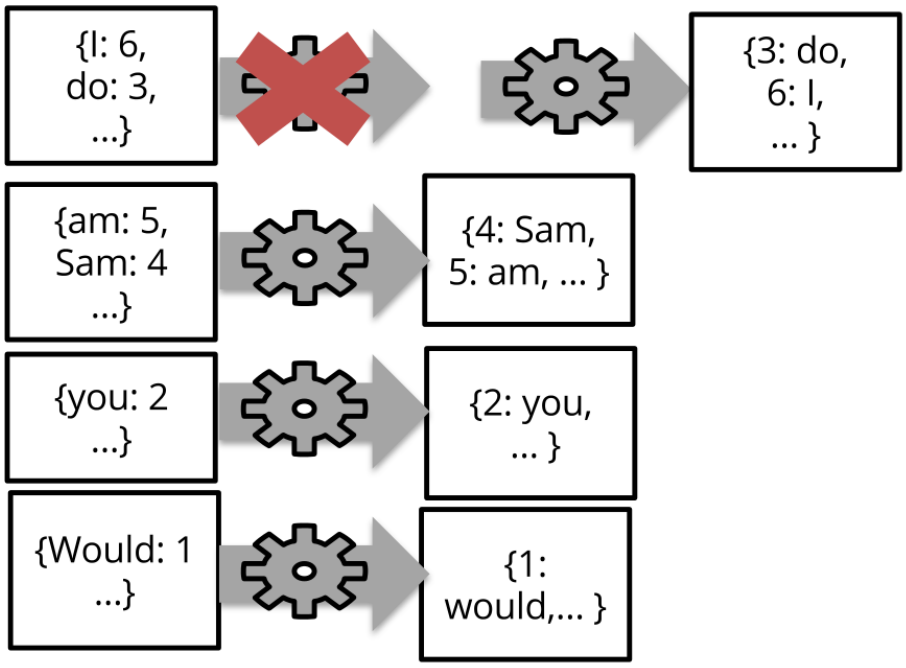


Figure 10.4: To deal with failure, we can just launch another task.

Large Parallel Databases did not support fault tolerance until recently. Therefore, if the query fails you have to redo it since the beginning. For example, if the query takes longer than 3 hours you will experience one failure on a server during the query. Thus, the query is useless.
How to deal with slow tasks? **Just launch another task!**

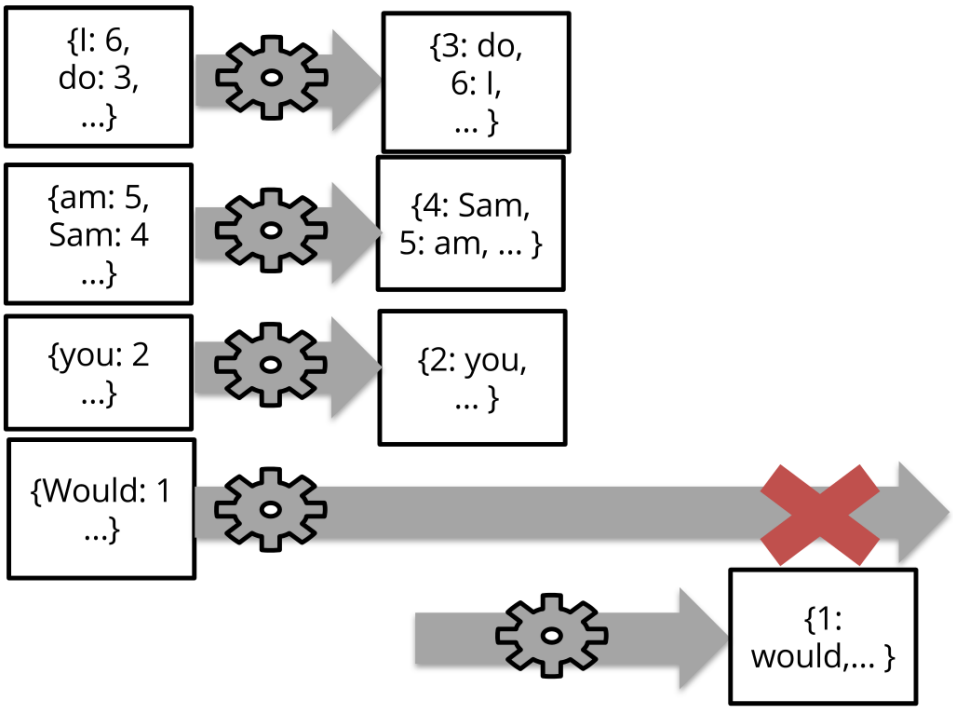


Figure 10.5: To deal with slow tasks, we can just launch another task.

Map-Reduce is not used by Google anymore, even if they invented it. More advance techniques exist nowadays. Why? Because Memory is really cheap! (RAM: 1 cent/Mb)

10.2 Spark Computing Framework

Spark provides a programming abstraction and parallel runtime to hide all of these complexity. Table 3 shows the main differences between Spark and Hadoop Map-Reduce. The big advantage with Spark is that it deals itself with all the complex clustering between the nodes.

	Hadoop Map Reduce	Apache Spark
Storage	Disk only	In-memory or on disk
Operations	Map and Reduce	Map, Reduce, Join, Sample, etc.
Ease of use	Java program	Scala and Python shells

Table 3: Main differences between Spark and Hadoop

One of the big advantage of Spark is that it uses in-memory storage. This can speed-up the computation time by a factor of 100 on the Logistic Regression for example. The good thing with Spark is that if the RAM is full, then it will use the memory on disk. We can see some interesting fact about having data **in-memory**:

- Optimized for batch, data-parallel ML algorithms
- An efficient, general-purpose language for cluster processing of big data
- In-memory query processing (Spark SQL)

Spark is not only good because it's faster than Hadoop. For example, we can see that some practical Challenges with Hadoop are:

- Very **low-level** programming model
- Very **little re-use** of Map-Reduce code between applications
- **Laborious** programming: design code, build jar, deploy on cluster
- **Relies heavily on Java reflection** to communicate with to-be-defined application code.

And some practical Advantages of Spark are:

- **High-level programming model**: can be used like SQL (Dataframe) or like a tuple store.
- **Interactivity**
- **Integrated UDFs** (User-Defined Functions)
- High-level model (**Scala Actors**) for distributed programming
- **Scala generics** instead of reflection: Spark code is generic over [Key, Value] types.

One last advantage of Spark is the way it deal with **Fault tolerance**. Hadoop uses data replication on HDFS. Therefore, once it's computed, you should not loose it. On the other hand, Spark remember **how** to recompute everything. For example, it will remember which nodes failed and which road he used to compute everything.

10.2.1 Read more

If you want to read more on Haddop, you can go on Apache Hadoop website. For Spark, you can go on Aparche Spark website. A good article comparing Spark and Hadoop can be found on Xplenty. You can always find more information about Spark and Hadoop on Internet. But the thing you should know is that even if Spark is much faster and much easier to use, Hadoop is not ready yet for the *elephant's graveyard*.

11 Handling Text

Again, this section does not have ambition to explicit all text handling methods but to give an overview of what are the difficulties encountered in this field.
Information Retrieval (IR) is **finding material** (usually documents) of an **unstructured nature** (usually text) that satisfies an **information need** from within **large collections** (usually stored on computers).
We name **collection** or **corpus** a set of documents

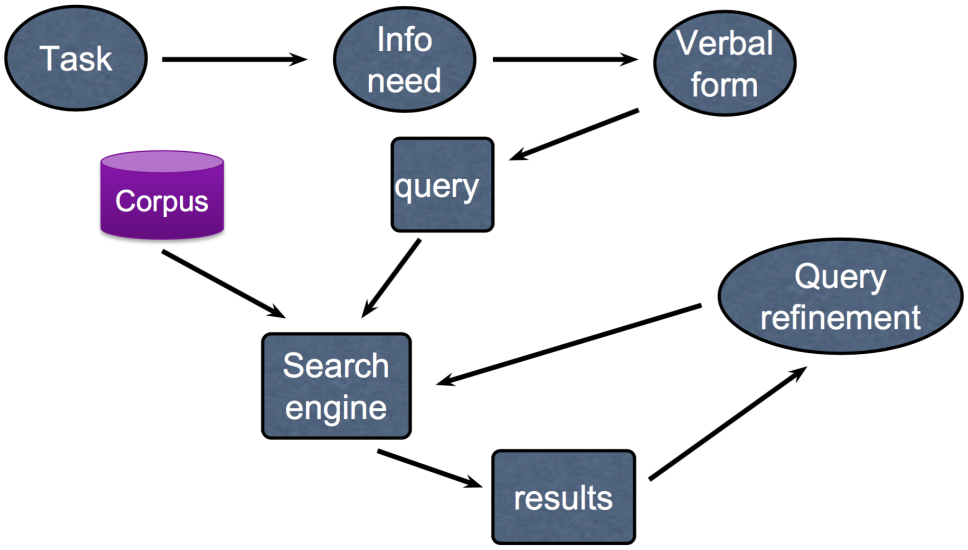


Figure 11.1: Search engines pipeline.

11.1 Types of Search Engines

- Q&A engines
- Collaborative
- Enterprise
- Web
- Metasearch
- Semantic
- NLP
- ...

11.2 Challenges of Information Retrieval

Important thing to always remember when dealing with information retrieval is that the main bottleneck is due to **human cognition** and not computational problems. People often have very unclear definition of concepts, even of very common ones, and they can often disagree between them.

11.2.1 Queries

Queries are (or can be) often submitted by users who do not know how works search engines and cannot help them to clarify ambiguities. Also, some users do not even know what they want to retrieve or how to name it.

11.2.2 The concept of *relevance*

Search engines try to retrieve **relevant** documents according to user queries. But an intrinsic problem it to define **what is relevant** because it changes from one user to another. Is relevance about usefulness? Novelties ? Interest? All of them?
At the end, only real people can judge efficiency of a search engine !

11.2.3 Representation

Computers need a way to represent the items contained in their database. How do you represent a document? An image? A video?
Typically, a document can be represented by a bag-of-words containing the most ones appearing inside it. But that's not enough. Someone can be more interested in meta-data such as the date of publication or author.

11.2.4 Semantic

Computers can handle words efficiently but, again, it is not sufficient. Meaning of the same word can differ from context to another, or even from opinion to another!
A *jaguar* could represent an animal or a car.
A *bank* could represent a financial institution, a blood stock or even... a school a fish.

11.3 Document views

- **Content view** is concerned with representing the content of the document; that is, what is the document about.
- **Data view** is concerned with factual data associated with the document (e.g. author names, publishing date)
- **Layout view** is concerned with how documents are displayed to the users; this view is related to user interface and visualization issues.
- **Structure view** is concerned with the logical structure of the document (e.g. a book being composed of chapters, themselves composed of sections, etc.)

11.4 NLP Pipeline

Here is described the pipeline followed when trying to represent a human-written document into an item representable by a computer.

- Parsing: Extract information from the document
  - regarding formats: html, PDF, excel, ...
  - character encoding: UFT-8, ...
  - languages

- Tokenization: token is an instance of a sequence of characters. All of them will be (after further processing) candidate to be the index entry of documents.
  - Pair of words going together: Hewlett-Packard must not be split during tokenization
  - Direction of reading: Arabic right to left, but left to right for numbers
  - Word Separator: Chinese and Japanese have no spaces between words
  - Alphabet: Japanese uses Katakana, Hiragana, Kani and Romaji alphabet mixture
- Stop-words list: remove all noisy words using pre-constructed (or enhanced) list.
  - english stop-words: and, to, or, the, ...
- Normalization
  - Case folding: reduce to lowercase, but can bring problems. *CAT* (acronym of Caterpillar Inc.) must not be mistaken with a *cat*.
  - Accent in French: *résumé* or *resume*
  - Umlaut in German: *Tuebingen* or *Tübingen*
- Thesauri and soundex link words with their synonyms
  - by hand-constructing equivalence classes: car = automobile
  - by expanding submitted queries: when search for *automobile* add it *car*
- Stemming / Lemmatization: normalizes the words to their stem or lemma form.
  - Stemming: reduce to the *root* by affix chopping (Porter's algorithm): example → exampl
  - Lemmatization: retrieve the canonical form of each word. More proper than stemming but also more complicated to use because must be based on a dictionary.

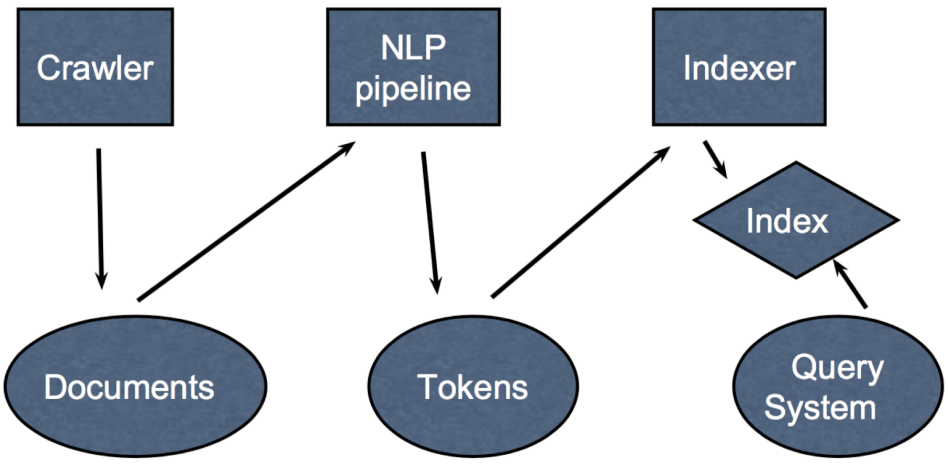


Figure 11.2: Search engines architecture.