Introduction to Data Science

WS 23/24, RWTH Aachen

February 2, 2024

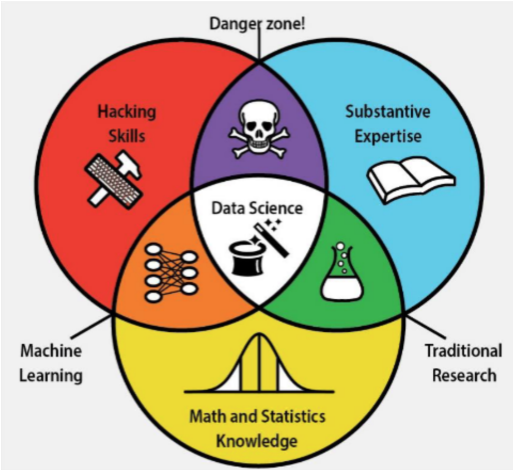
Contents

Introduction

Let's first introduce the general term of data science. It is a new and important discipline that can be viewed as an amalgamation of classical disciplines such as statistics, data mining, databases, and distributed systems, with additional new challenges constantly emerging and making the field highly dynamic and appealing.

The problems grow in terms of size ("Big data") and complexity of the questions to be answered. But the basic job can be summarized as:

- Basic job: Input (data) → Processed by data scientist (with tools) → Output (value)
- Where the skills of a data scientist are the combination of an open mind, human interest, analytical skills, creativity, business-benefiting weighting, etc.
- Or in other terms as can be seen in 0.1



- Data science: due to its interdisciplinary nature requires an intersection of the following abilities (H + M + S)
- Hacking skills (H): necessary for working with massive amounts of electronic data that must be acquired cleaned, and manipulated
- Math and statistics knowledge (M): to choose appropriate methods and tools to extract insight from data
- Substantive expertise in a scientific field (S): crucial for generating motivating questions and hypotheses and interpreting the results
- Traditional research: in the intersection of M and S
- Machine learning: combines H with M, but doesn't require scientific motivation
- !Danger zone: H and S combined without rigorous methods can beget incorrect analyses

Figure 0.1: Skillset of a data scientist

With the growing importance of data and digitalization, organizations are looking for data scientists, who may outnumber computer scientists in the future. Important is the ability to handle data in any form, so basically the need for an all-around skilled "data wizard". This importance can be further highlighted when looking at the tech development over the past 20 to 30 years. While the hardware got tremendously cheaper, faster, and more compact (20 times faster for MIP = mixed integer programs), also

software has progressed in terms of speed (50 times faster for MIP). Interesting to look at is also the aspect of automation.

Dimensions of data science are:

- The different types of data (structured or unstructured, text, images, events, ...)
- The different types of tasks (supervised or unsupervised, ...)
- Human versus machine (Who does what?)
- Algorithm versus visualization (What is needed?)
- Flexibility versus usability
- Scalability versus quality (exact versus heuristics)
- Responsibility versus utility (accuracy and precision versus fairness, privacy, transparency, ...)

Besides raw data science, interesting to look at is also the connection to process science. The interplay between process and data science (PADS) leads to the term process mining. Imagine the connection as shown in 0.2.

Figure 0.2: Interplay between process science and data science

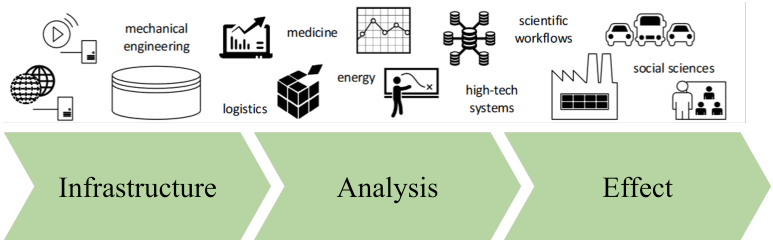
As the final part of the introduction, we will now see the general covered topics in this course:

- Basic data exploration and visualization
- Decision trees, regression, support vector machines
- Neural networks, evaluation of supervised learning problems, clustering
- Frequent items sets, association rules, sequence mining, process mining, text mining
- Data preprocessing/quality and binning, visual analytics and information visualization
- Responsible data science
- Big data technologies

1 Basics of data science

1.1 Data science pipeline

First, we are going to look at how data is processed in terms of the **data science pipeline** as it can be seen in 1.1.



Data science pipeline

Figure 1.1: Pipeline of data science

Let's look at the individual components. The first step to pay attention to when wanting to handle data is the **infrastructure** with the keywords **"volume and velocity"**. The main challenge is making things scalable and instant (responsiveness). Important terms are for example:

Infrastructure

- Instrumentation
- Big data infrastructures, distributed systems
- Data engineering (databases and data management)
- Programming
- Security

Next, we have the step of the actual **analysis** concerned with **extracting knowledge** from data. The core challenge can be put as providing answers to known and unknown unknowns. Important terms are for example:

Analysis

- Statistics, algorithms
- Data and process mining
- Machine learning, artificial intelligence
- Operations research
- Visualization

Finally, we also need to be concerned with the **effect** of our results on people, organizations, and society. The main challenge of this pipeline step is to do **responsibly** perform data handling. Important terms are for example:

Effects

- Ethics and privacy, and IT law
- Human-technology interaction
- Operations management
- Business models, entrepreneurship

This course will look into all the steps of the pipeline, but the main focus lies on the data analysis.

Four generic data science questions

The questions vary in their difficulty and prediction into the future:

1. **What** happened?
2. **Why** did it happen?
3. What will happen in the **future**?
4. What is the **best** that can happen?

Important while answering these questions is to keep attention to all three pipeline steps, so not only what analysis we need to perform to answer them, but also how we collect our input (data) and how to deal with our output (result).

1.2 Types of data

Now that we know that we have some kind of data as our input, we need to take a look at what this data can look like. Generally speaking, there are two types:

Structured data
- Structured data like age, time, gender, class, etc., and

Unstructured data
- Unstructured data like text, audio, video, etc.

For **structured data** we have a further subdivision into structured data types. The data types depicted in 1.2 will be described in detail.

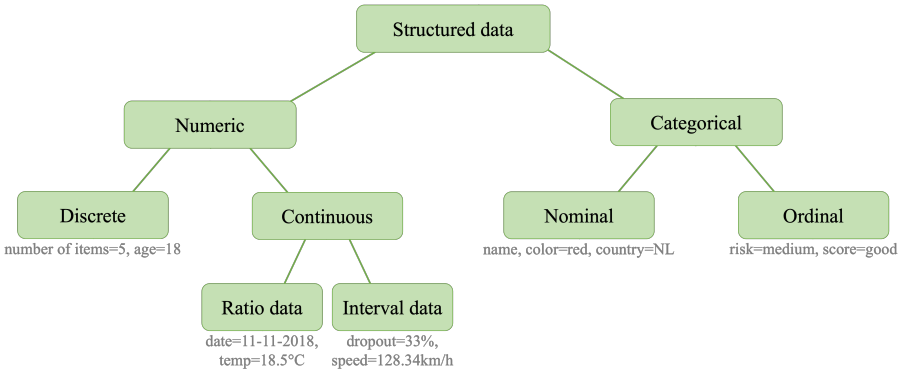


Figure 1.2: Overview structured data types

Categorical data
- **Categorical** data can be stored and identified based on names or labels given to them and is also known as "qualitative" data. Matching can be applied, where data is grouped based on similarities.

Nominal data
- Concretely, **nominal** data or naming data has a label and its characteristic similar to a noun and doesn't imply an order.

Ordinal data
- **Ordinal** data on the other hand is ranked, ordered, or used on a rating scale. This means, you can count and order ordinal data but are not able to measure it.

Numerical data
- In contrast to categorical data, we also have **numerical** data referring to data in the form of numbers instead of another language or descriptive form. It is also known as "quantitative" data. Important is the ability to be statistically and arithmetically calculated (allowing for $+, -, >, =, \ldots$).

Discrete data
- One subtype of numerical data is **discrete** data representing countable items, that are collected in a list (finite or infinite).

- Then, there's also **continuous** data in the form of intervals or ranges. The data represents measurements with their intervals falling on a number line (so counting isn't involved). Continuous data
- Continuous data can now be further distinguished. One subtype is **interval** data where the data can be measured only along a scale at equal distances from each other, so only addition and subtraction operations are allowed. There is no true zero (and hence no $\cdot, /$). Interval data
- And finally, we have **ratio** data describing measurement with a defined (true) zero point. Ratio data

For **unstructured data**, we just take the raw data and interpret it as a stream of bits. This goes for text, audio, images, signals, and videos exactly the same. Examples can be seen in 1.3.

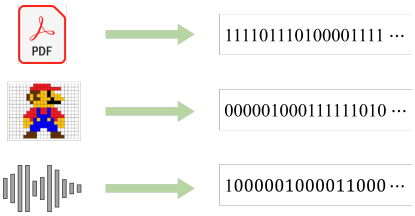


Figure 1.3: Input for unstructured data

Data can now be stored and ordered together by putting it into **tables**. Concretely, columns represent different features (can be different kinds of data types) whereas rows describe data instances (also known as individuals, entities, cases, objects, or records). Examples can be seen in 1.4. Tabular data

feature

Order id	Product	Price	Date	From	To	Message	Image
32424	718 Cayman	66.000	21-10-2018	Sue	Peter	"How are you?"	☺
34535	911 Carrera	102.000	22-10-2018	Peter	Sue	"Very good!"	
43555	911 Turbo	154.000	24-10-2018	Peter	Mary	"Let's go out."	
...	...	...	...	...	...	...	...
Ordinal	Nominal	Ratio data	Interval data	Nominal			Unstructured

instance

Figure 1.4: Table data with data types

Features can now be raw or derived (e.g. max, min, average, rank, bin, ...). An important aspect is time, as it cannot decrease and we usually want to predict the future based on the past. Features

An important distinction to be made when it comes to tabular data is whether the items are labeled or not.

- In case of labelled data we have **descriptive features** and a **target feature**. Labelled data
  - The descriptive features are also known as predictor variables or independent variables. Descriptive features

Target feature

  - Alternative names for target features are response variable, dependent variable, or also label.

Unlabelled data

- Unlabelled data on the other hand doesn't have a selected target feature.

1.3 Supervised and unsupervised learning

Derived from the different kinds of tabular data, we have two fundamental learning paradigms. Exemplary input data and possible results for both paradigms can be seen in 1.5.

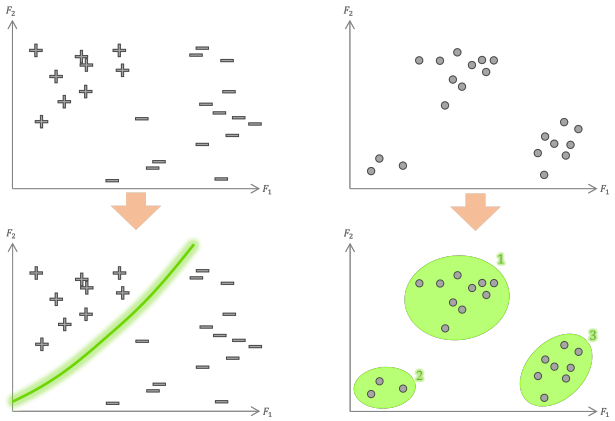


Figure 1.5: Comparing supervised (left) and unsupervised (right) learning

Supervised learning

In the case of labeled data, we can apply **supervised learning**. The goal is to find a "rule" in terms of descriptive features explaining the target feature as well as possible. Examples include:

- Hospital environments where the target variable can be "recover" (yes or no), and the descriptive variables can be age, gender, smoking, ...
- University environments where the target variable can be "drops out" (yes or no), and the descriptive variables can be mentor, prior education, ...
- Production environments where the target variable can be "order is delivered in time" (yes or no), and the descriptive variables can be product, agent, ...

Unsupervised learning

In contrast to labeled data, we can also have instances without target labels, where we can only apply techniques of **unsupervised learning**. The goal is to find clusters or patterns.

Cluster

- **Clusters** are homogeneous sets of instances. Examples include finding similar groups of patients, students, customers, orders, cars, companies, and so on.

Pattern

- **Patterns** on the other hand reveal hidden structures in the data, so basically the unknown unknowns. Rules of some form can be found in many environments. Examples can look like:
  - Customers who buy bread and butter typically pay by phone.
  - Patients who drink and smoke typically pay the hospital bill earlier than others.
  - Products produced by team A on Monday tend to be returned more frequently by customers.

Interesting to regard is process discovery as a form of unsupervised learning in the way that a process model is just a very sophisticated rule. Important to mention, that this task can get very complex very quickly.

Terminology

Important to see for all of data science: many different names are used to refer to the key disciplines contributing to data science.

- This includes statistics, data analytics, data mining, machine learning, artificial intelligence, predictive analytics, process mining, generative AI, etc.
- Since frequently the same name is used for different concepts (names describe heavily overlapping areas), they really need to be put in context and interpreted accordingly.

The point can be highlighted when looking at scoping machine learning. Here are examples of confusions:

- Sometimes "machine learning" is used as a synonym for "deep neural networks" and sometimes they cover the entire spectrum of learning techniques.
- Neural networks can be used as classifiers. But this doesn't imply that numerous classification techniques developed in data mining are part of machine learning in the narrow sense.

How confusing specifically the arrangement of terms around machine learning is and how fluent and unclear the actual terms are, is depicted in 1.6.

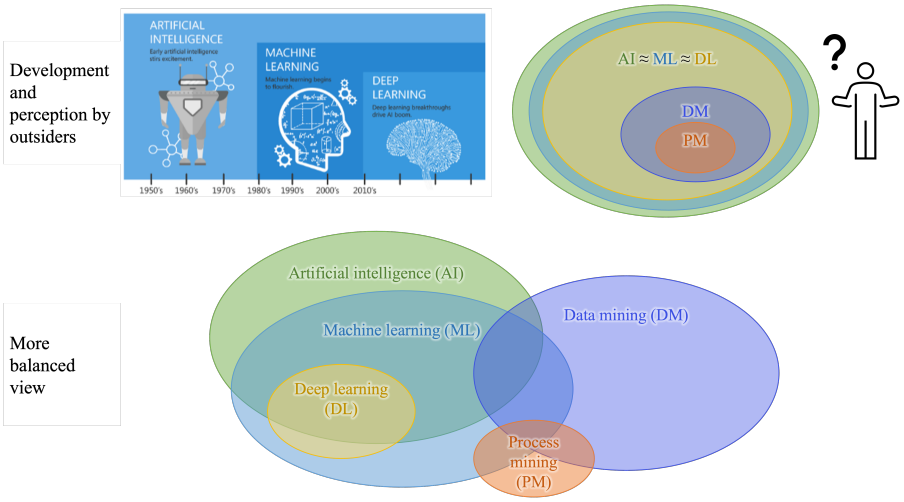


Figure 1.6: Terms around machine learning

1.4 Data science process

There are many different lifecycle models to describe phases in a data science project. This section will give a quick overview of some important ones.

CRISP-DM

We'll start with **CRISP-DM** which stands for "Cross-industry standard process for data mining". It was developed in the late 1990s by different involved companies (SPSS, Teradata, Daimler AG, NCR Corporation, Ohra). The process consists of multiple steps playing together as visualized in 1.7

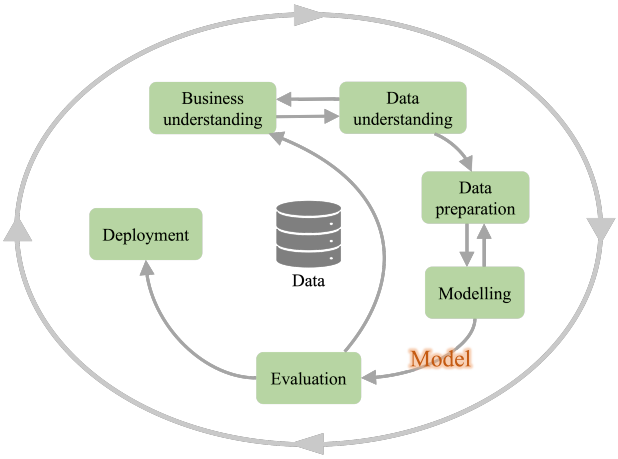


Figure 1.7: CRISP-DM process

Business understanding

Determine business objective	Background, business objective, business success criteria
Situation assessment	Inventory of resources, requirements, assumptions, constraints, risks, contingencies, terminology, costs, benefits
Determine data mining goal	Data mining goals, data mining success criteria
Produce project plan	Project plan, initial assessment of tools and techniques

Data understanding

Collect initial data	Initial data collection report
Describe and explore data	Data description, exploration report
Verify data quality	Data quality report

Data preparation with starting point: data set (with description)

Select data	Rationale for inclusion and exclusion
Clean data	Data cleaning report
Construct data	Derived attributes, generated records
Integrate and format data	Merged/reformatted data

Modeling[1]

Select modeling technique	Modeling technique, modeling assumptions
Generate test design	Test design
Build model	Parameter settings, models, model description
Assess model	Model assessment, revised parameter settings

Evaluation

Evaluate results	Assessment of data mining results w.r.t. business success criteria, approved models
Review process	Review of process
Determine next steps	List of possible actions settings

Deployment

Plan deployment	Deployment plan

[1] The term "modeling" can be misleading. Meant is the selection and assumptions by a human, or automated learning by a tool or algorithm

Plan monitoring, maintenance	Monitoring and maintenance plan
Produce final report	Final report and final presentation
Review project	Experience documentation

Next, we have the **KDD** (Knowledge Discovery in Databases) process as shown in 1.8. Another process model also developed by SAS institute is called **SEMMA** consisting of the phases Sample, Explore, Modify, Model, and Assess.

KDD
SEMMA

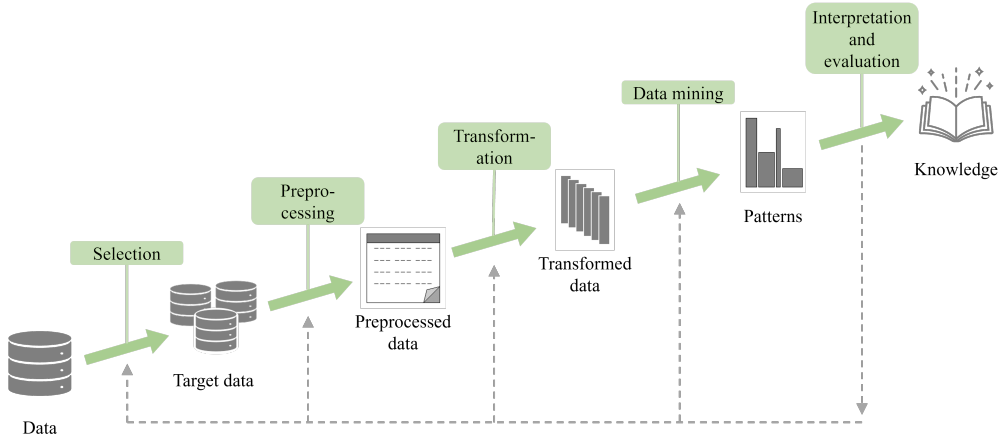


Figure 1.8: KDD process

The next process model is specifically developed for **L* lifecycle model** with multiple stages as shown in 1.9.

L* lifecycle model

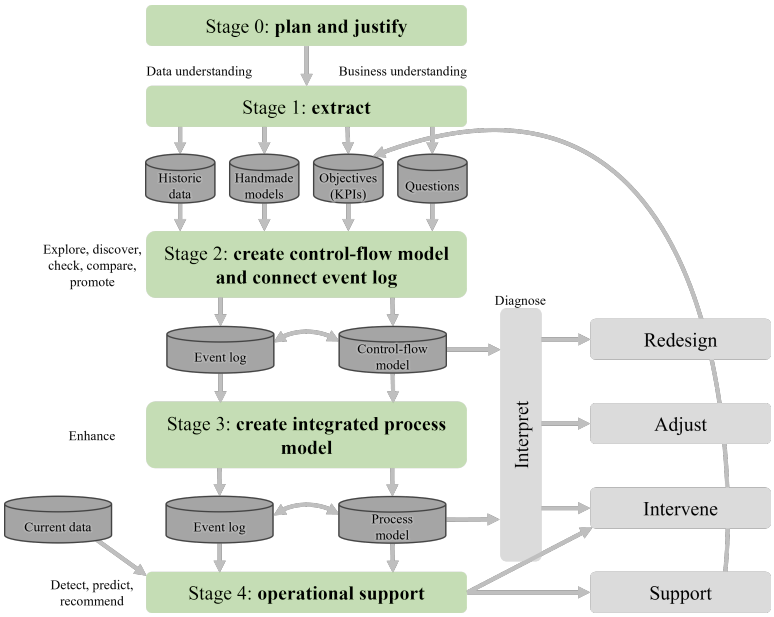


Figure 1.9: L* lifecycle model

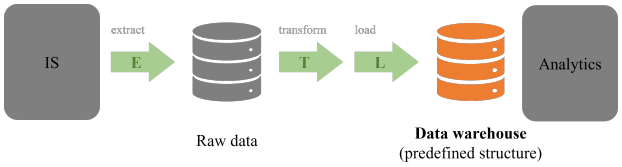
Furthermore, we have two methodologies the process model can be related to. Important

to implement and solidify are improvements in both.

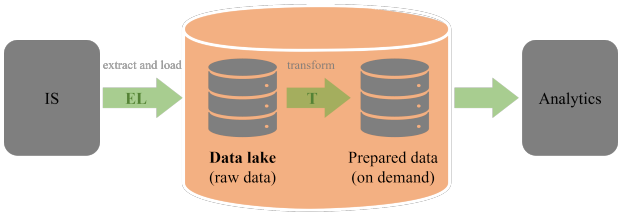
PDCA
- **PDCA** stands for Plan-Do-Check-Act and is a never-ending cycle with exactly these steps.

DMAIC
- The other one **DMAIC** stands for Define-Measure-Analyze-Improve-Control, with the following subtasks:
  - Define: launch team, establish charter, plan project, gather VOC/VOB, plan for change
  - Measure: document process, collect baseline data, narrow project focus
  - Analyze: analyze data, identify root causes, identify and remove waste
  - Improve: generate, evaluate, and optimize solutions, pilot, plan and implement
  - Control: control the process, validate project benefits

ETL
ELT
Finally, we have two processes with the same components, but different ordering of the steps as can be seen in 1.10. The short terms for the processes are **ETL** (extract, transform, load) and **ELT** (extract, load, transform).



(a) ETL with data warehouse



(b) ELT with data lake

Figure 1.10: Processes with extraction, transform, and load steps

80/20 rule

As a final note on which steps are usually the most time-expensive: there is a so-called "80/20 rule" stating:

- 80% of a data scientist's time is spent on finding, cleaning, preprocessing, and organizing data. This leaves only 20% to actually perform an analysis.
- On the other hand, we have 20% effort determining 80% of the final result.

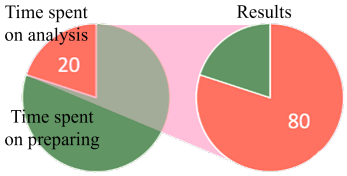


Figure 1.11: 80-20 rule

1.5 Challenges

To finalize the overview and basics of data science, let's look at the typical challenges.

First, we have the challenge of **finding data**. There may be hundreds or thousands of tables (E.g., in the case of SAP the numbers can easily for up to 800′000). But, different entities differ in their relevance, meaning some are less relevant than others.

The next challenge is the **transformation of data**, meaning reorganization of data, filtering, extraction of relevant features, and so on. Not only for transformations, but also in general other challenges are **dealing with big data and streaming data**. The challenge of big data evolved over the last few decades, meaning

- Typical stochastic methods try to solve the problem of saying something about entities given only a small amount of samples, whereas
- Now we have a very high load of data and need to solve the problem of dealing with these large amounts in a correct way.

Also for streaming data, new approaches need to be thought of. Additionally, we also need to **deal with a concept shift**.

Another huge challenge is ensuring **data quality**. This goes especially, since our provided data may be incomplete, invalid, inconsistent, imprecise, and/or outdated. Consider for example timestamps. They might be

- Incomplete (event is missing),
- Invalid (e.g. 14-14-2018),
- Inconsistent (14-07-2018 in contrast to 7-14-2018), or
- Imprecise (only regard part of available data: 2018-09-21~~'T'13:00:10~~).

A very typical problem is **overfitting and underfitting** as it can be seen 1.12.

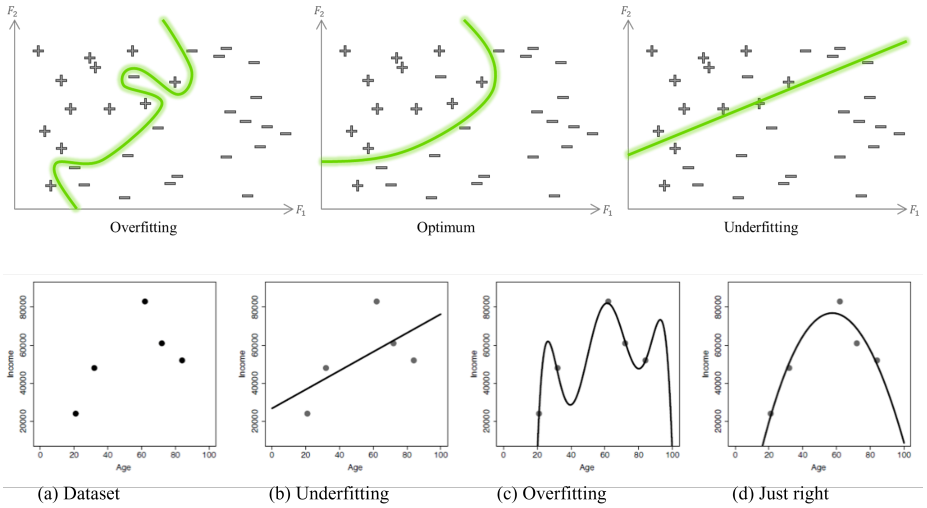


Figure 1.12: Over- and underfitting visualized

The next challenge is the distinction of **correlation and causation**, explicitly that correlation does not imply causation. Consider this example:

- Sunburn and ice cream have a strong correlation. When only these two features are considered, one might derive that either ice cream causes sunburn, or the other way around.
- We know of course, that this is not correct and instead an additional factor causes both phenomena: if the sun is shining, it's warm and people eat ice cream, and also sun directly causes sunburn.

Besides the accuracy of our results, we also need to look into whether our results are valuable. Concretely, **results** should be **made actionable**. This means, that analysis results should be relevant, specific, timely, novel, and clear. Our goal is to go from "data" to "insight" and finally "action". Consider these examples:

- Warning about a traffic jam should come before entering said traffic jam.
- That it's currently raining is not too helpful information. Preferably is a notice ahead of time.

Responsible data science

The last, but very important challenge is **responsible data science** (RDS). This includes ensuring of:

- **Fairness**, meaning data science should exclude prejudice (How to avoid unfair conclusions even if they are true?)
- **Accuracy**, so data science without guesswork (How to answer questions with a guaranteed level of accuracy?)
- **Confidentiality** (How to answer questions without revealing secrets?)
- **Transparency** (How to clarify answers such that they become indisputable?)

2 Data visualization and exploration

2.1 Data extraction

Generally, we have a bunch of different data sources, with a multitude of different standards, features, and also information that can be extracted. The general problem is depicted in 2.1.

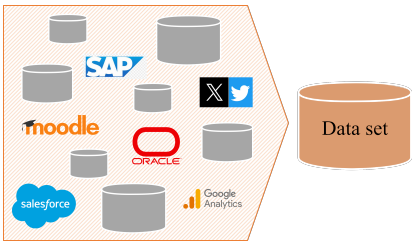


Figure 2.1: Data extraction from different sources

The different datatypes were already analyzed in the previous chapter, but still 2.2 shows a quick recap. Important to mention, that any unstructured data is considered a bit stream.

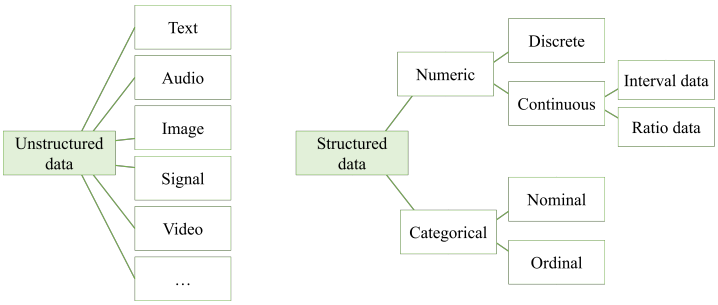


Figure 2.2: Recap: overview types of data

Important when wanting to obtain any object is of course the feature extraction. The data described by features are usually captured in a tabular form, with rows as the instances and columns as the features. There exist some special features:

- **Time** usually always plays a role in data observation, which is why it is usually one of the recorded features.
- Then there are also the **target features**, in contrast to the descriptive features. The concept was introduced in the last section as part of supervised learning.

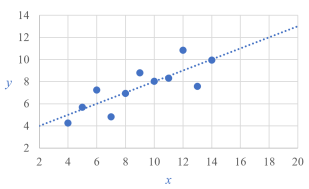
Importance of visualization

We now looked at how we can represent our data in a very machine-friendly represented way. The following subsection shows, why visualization of data has any importance, even though tabular data already captures the features nicely.

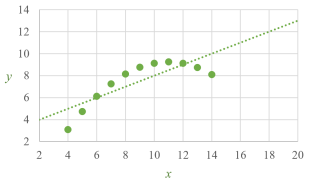
It is important as a human to explore your data before applying mathematical operations to see, which techniques make sense to apply to the provided data. As an extreme example, we will take a look at **Anscombe's quartet** created by Francis Anscombe in 1973.

	I		II		III		IV	
	x	y	x	y	x	y	x	y
	10	8.04	10	9.14	10	7.46	8	6.58
	8	6.95	8	8.14	8	6.77	8	5.76
	13	7.58	13	8.74	13	12.74	8	7.71
	9	8.81	9	8.77	9	7.11	8	8.84
	11	8.33	11	9.26	11	7.81	8	8.47
	14	9.96	14	8.1	14	8.84	8	7.04
	6	7.24	6	6.13	6	6.08	8	5.25
	4	4.26	4	3.1	4	5.39	19	12.5
	12	10.84	12	9.13	12	8.15	8	5.56
	7	4.82	7	7.26	7	6.42	8	7.91
	5	5.68	5	4.74	5	5.73	8	6.89
Mean of x	9		9		9		9	
Sample variance of x	11		11		11		11	
Mean of y	7.5		7.5		7.5		7.5	
Sample variance of y	4.125		4.125		4.125		4.125	
Correlation between x / y	0.816		0.816		0.816		0.816	
Linear regression line $y(x)$	$3 + 0.5\,x$		$3 + 0.5\,x$		$3 + 0.5\,x$		$3 + 0.5\,x$	

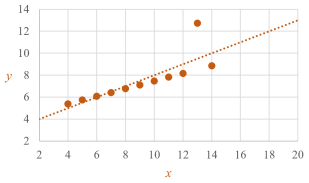
Data set I



Data set II



Data set III



Data set IV

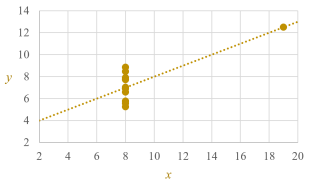


Figure 2.3: Anscombes quartet

- You can see the raw data of all four datasets in the table in 2.3. Since the format isn't human-friendly to read, you might not see any significant differences in the data.
- Now consider applying the evaluation depicted below the data table. As you can see: all of the properties that are evaluated are exactly the same for all datasets.
- BUT, if you visualize the datasets, e.g. as simple scatter plots, you see, how drastically they vary. These show the importance of first exploring your data, to then have a better evaluation foundation for the applied techniques.

The next example highlighting the importance of visualization and especially of a fitting and well-thought-out visualizationis the diagram as shown in 2.4.

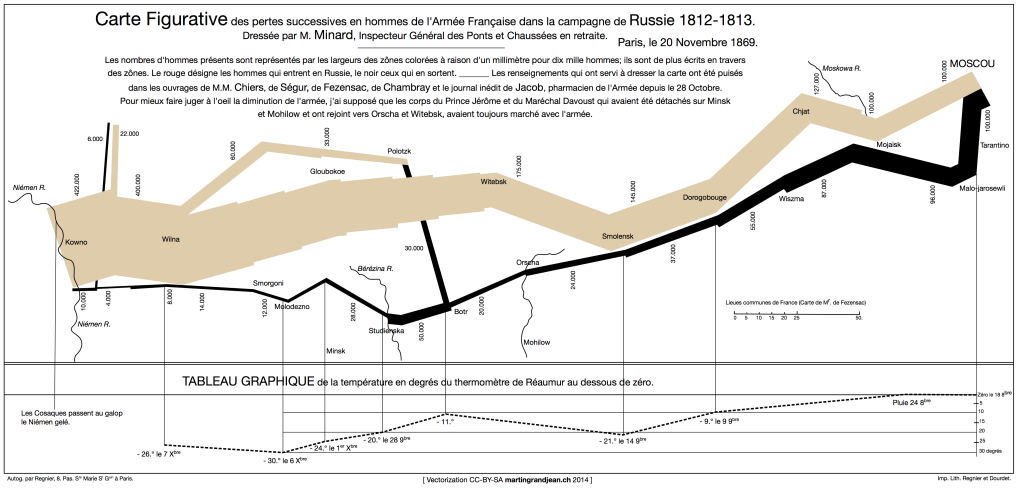


Figure 2.4: Multi-feature visualization (Napoleon's army)

The chart shows the following aspects, which are quite a lot, while still keeping a good overview:

- The number of men in Napoleon's 1812 Russian campaign army,
- Their movements (direction),
- The temperature encountered on the return path,
- All given a specific geographic point.

The final example highlights the importance of visualizing event or process data. In 2.5 we see the plot of different given event data input. With the visualization, some sort of trend, similarities, certain batching areas, etc. can be directly seen.

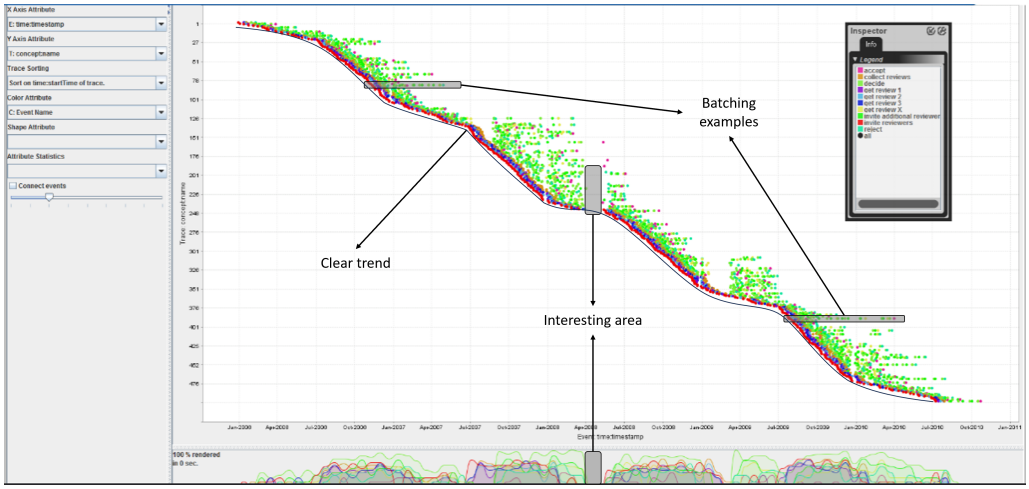


Figure 2.5: Visualization of event data

2.2 Characterizing individual features

As a first step of actual data exploration, we are now going to look at which information we can get from a single data feature. In terms of tabular data: we're going to focus on a single column.

What kind of data we can derive from the feature, depends of course on the data type. Generally deriving features from other ones is of course done best when dealing with structured data. Here, we have two types from which we can derive different properties.

Investigation of individual **continuous** features

From **continuous features**, we can derive

- count : Number of instances having this feature
- % miss : Percentage of missing information (how many instances don't have this feature)
- card : Number of unique values (cardinality)
- min : Minimal value over all instances
- 1^{st} qrt : 25^{th} percentile (largest value of the quarter of instances having the lowest values)
- mean : Average value over all instances
- median : Middle value of all instances
- 3^{rd} qrt : 75^{th} percentile (smallest value of the quarter of instances having the highest values)
- max : Maximal value over all instances
- std. dev : Standard deviation over all instances

Investigation of individual **categorical** feature

From **categorical features**, we can derive[2]

- count : Number of instances having this feature
- % miss : Percentage of missing information (how many instances don't have this feature)
- card : Number of unique values (cardinality)
- mode : Most common value
- mode frequ : Frequency of the mode
- mode % : Percentage of the mode
- 2^{nd} mode : Second most common value
- 2^{nd} mode frequ : Frequency for the second mode
- 2^{nd} mode % : Percentage of the second mode

To get a better idea of how to get these properties for all of the features, we will look at an example. Consider a table containing information about insurance claims fraud. The dataset contains 500 instances (claims) and a bunch of different features such as type, claim amount, etc. Now first, determine the data type of each feature, and then

[2] obvious: min, max, mean, etc. can't be computed

create one table for the numerical and one for the categorical features and fill it with the according information. The raw data can be found in 2.6.

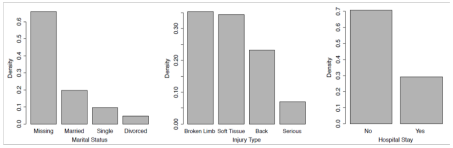
ID	Type	Inc.	Marital Status	Num Clmnts.	Injury Type	Hospital Stay	Claim Amnt.	Total Claimed	Num Claims	Num Soft Tiss.	% Soft Tiss.	Claim Amt Rcvd.	Fraud Flag
1	CI	0		2	Soft Tissue	No	1,625	3250	2	2	1.0	0	1
2	CI	0		2	Back	Yes	15,028	60,112	1		0	15,028	0
3	CI	54,613	Married	1	Broken Limb	No	-99,999	0	0	0	0	572	0
4	CI	0		4	Broken Limb	Yes	5,097	11,661	1	1	1.0	7,864	0
5	CI	0		4	Soft Tissue	No	8869	0	0	0	0	0	1
6	CI	0		1	Broken Limb	Yes	17,480	0	0	0	0	17,480	0
7	CI	52,567	Single	3	Broken Limb	No	3,017	18,102	2	1	0.5	0	1
8	CI	0		2	Back	Yes	7463	0	0	0	0	7,463	0
9	CI	0		1	Soft Tissue	No	2,067	0	0	0	0	2,067	0
10	CI	42,300	Married	4	Back	No	2,260	0	0	0	0	2,260	0
⋮													
300	CI	0		2	Broken Limb	No	2,244	0	0	0	0	2,244	0
301	CI	0		1	Broken Limb	No	1,627	92,283	3	0	0	1,627	0
302	CI	0		3	Serious	Yes	270,200	0	0	0	0	270,200	0
303	CI	0		1	Soft Tissue	No	7,668	92,806	3	0	0	7,668	0
304	CI	46,365	Married	1	Back	No	3,217	0	0		0	1,653	0
⋮													
458	CI	48,176	Married	3	Soft Tissue	Yes	4,653	8,203	1	0	0	4,653	0
459	CI	0		1	Soft Tissue	Yes	881	51,245	3	0	0	0	1
460	CI	0		3	Back	No	8,688	729,792	56	5	0.08	8,688	0
461	CI	47,371	Divorced	1	Broken Limb	Yes	3,194	11,668	1	0	0	3,194	0
462	CI	0		1	Soft Tissue	No	6,821	0	0	0	0	0	1
⋮													
491	CI	40,204	Single	1	Back	No	75,748	11,116	1	0	0	0	1
492	CI	0		1	Broken Limb	No	6,172	6,041	1		0	6,172	0
493	CI	0		1	Soft Tissue	Yes	2,569	20,055	1	0	0	2,569	0
494	CI	31,951	Married	1	Broken Limb	No	5,227	22,095	1	0	0	5,227	0
495	CI	0		2	Back	No	3,813	9,882	3	0	0	0	1
496	CI	0		1	Soft Tissue	No	2,118	0	0	0	0	0	1
497	CI	29,280	Married	4	Broken Limb	Yes	3,199	0	0	0	0	0	1
498	CI	0		1	Broken Limb	Yes	32,469	0	0	0	0	16,763	0
499	CI	46,683	Married	1	Broken Limb	No	179,448	0	0		0	179,448	0
500	CI	0		1	Broken Limb	No	8,259	0	0	0	0	0	1

Figure 2.6: Example for single feature investigation (insurance claim fraud)

To investigate the raw data further, let's first extract the resulting feature-describing tables and then also visualize the data. More specifically for the visualization, we're going to show the distributions of the different features. Figure 2.7 shows both the properties of the features and examplary plots.

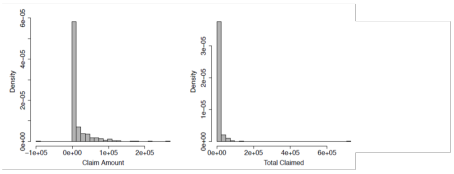
- For finite amounts of possible feature classes, simply visualize the distribution as a bar diagram with the different classes as entries on the x-axis. The y-axis can either be the frequency or a percentage.
- For continuous features with continuous variables/infinitely many possible feature values, group items (**binning**) and then visualize the resulting histogram. Binning

Feature	Count	% Miss.	Card.	Mode	Mode Freq.	Mode %	2nd Mode	2nd Mode Freq.	2nd Mode %
Insurance Type	500	0.0	1	CI	500	100.0	–	–	–
Marital Status	500	61.2	4	Married	99	51.0	Single	48	24.7
Injury Type	500	0.0	4	Broken Limb	177	35.4	Soft Tissue	172	34.4
Hospital Stay	500	0.0	2	No	354	70.8	Yes	146	29.2



(a) Categorical features

Feature	Count	% Miss.	Card.	Min	1st Qrt.	Mean	Median	3rd Qrt.	Max	Std. Dev.
Income	500	0.0	171	0.0	0.0	13,740.0	0.0	33,918.5	71,284.0	20,081.5
Num Claimants	500	0.0	4	1.0	1.0	1.9	2	3.0	4.0	1.0
Claim Amount	500	0.0	493	-99,999	3,322.3	16,373.2	5,663.0	12,245.5	270,200.0	29,426.3
Total Claimed	500	0.0	235	0.0	0.0	9,597.2	0.0	11,282.8	729,792.0	35,655.7
Num Claims	500	0.0	7	0.0	0.0	0.8	0.0	1.0	56.0	2.7
Num Soft Tissue	500	2.0	6	0.0	0.0	0.2	0.0	0.0	5.0	0.6
% Soft Tissue	500	0.0	9	0.0	0.0	0.2	0.0	0.0	2.0	0.4
Amount Received	500	0.0	329	0.0	0.0	13,051.9	3,253.5	8,191.8	295,303.0	30,547.2
Fraud Flag	500	0.0	2	0.0	0.0	0.3	0.0	1.0	1.0	0.5



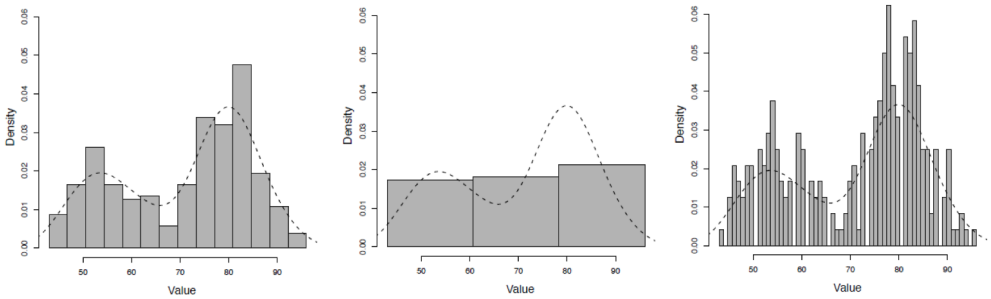
(b) Numerical (continuous) features

Figure 2.7: Feature-discribing table and distribution visualization

Over- and underfitting for binning

The binning comes with some challenges. When we select the amount of bins with evenly distributed width of each individual bin, we need to be aware not to **over- or underfit**. Examples can be found in 2.8. As one can see:

- In the case of underfitting, the true function is not at all matched.
- In the case of overfitting, there exist very steep valeys, the provided data points are more learned by heart rather than abstracting to a function.



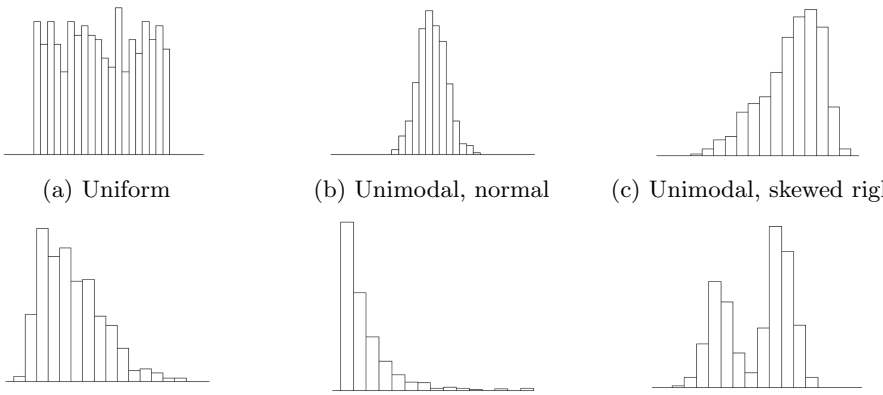
(a) Good approximation
14 bins

(b) Underfitting
3 bins

(c) Overfitting
60 bins

Figure 2.8: Binning for continuous variables

The histograms furthermore can show different types, as depicted in 2.9.



(a) Uniform

(b) Unimodal, normal

(c) Unimodal, skewed right

(d) Unimodal, skewed left

(e) Exponential

(f) Multimodal

Figure 2.9: Histogram types

Here are some further notes on the types:

Uniform
- **Uniform** means all items have the same likelyhood (within a range).

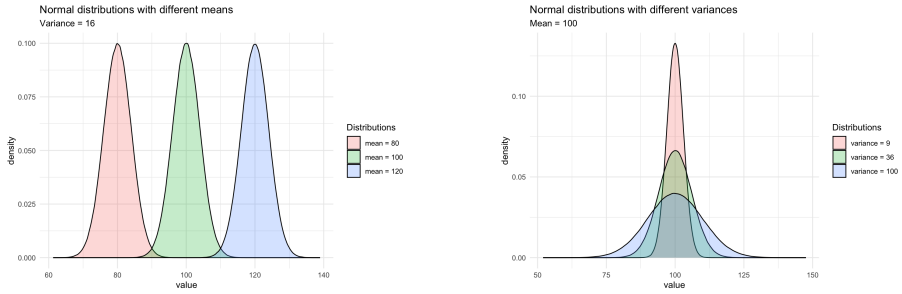
Unimodal Multimodal
- **Unimodal** means we have one peak (can be tilted to one side), whereas **multimodal** means there are multiple distinct ones.

Exponential
- **Exponential** means we have an exponential descrease in the likelihood over all instances.

Normal distribution

One of the types mentioned, we're now gonna investigate a bit further. The **normal distribution** is described by two important variables, whose effects on the distribution are shown in 2.10: Normal distribution

- The **mean**μ, so the expected value also characterizing the peak, and Mean
- The **standard deviation**σ characterizing how narrow the peak, or the distribution around the peak, is. Standard deviation



(a) Variation of mean μ (b) Variation of standard deviation σ

Figure 2.10: Normal distribution

The normal probability distribution over x is defined as:

$$x \sim \mathcal{N}(\mu, \sigma)$$
$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$$

Interesting are now precise areas we instantly know something about. The 68-95-99.7-rule tells us, as depicted in 2.11. 68-95-99.7-rule

- 68% of all observations will be within 1σ-distance of the mean,
- 95% of all observations will be within 2σ-distance of the mean, and
- 99.7% of all observations will be within 3σ-distance of the mean

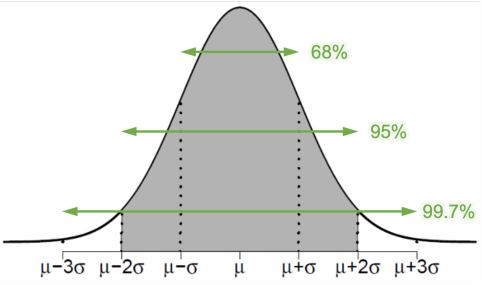
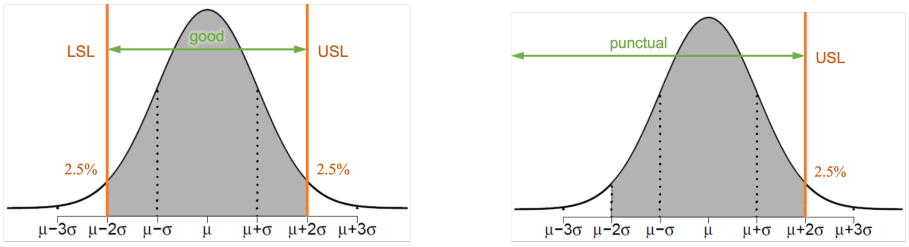


Figure 2.11: 68-95-99.7-rule

From this rule, we can now derive probabilities for different events. Consider the following examples, also visualized in 2.12:

- Example 1 is interested in the amount of defects for some produced item. The **tolerance** can be defined as within the 2σ-range, so with:
  - **LSL** (lower spcification limit) at $\mu - 2\sigma$, meaning only 2.5% of the instances have a larger deviation into the negative direction from the mean than this limit, and
  - **USL** (upper spcification limit) at $\mu - 2\sigma$, meaning only 2.5% of the instances have a larger deviation into the positive direction from the mean than this limit.

  Combined, $100\% - 2.5\% - 2.5\% = 95\%$ have a deviation from the mean lying withing the defined range.
- Example 2 is interested in how many deliveries are too late. Therefore, it **only** defined an **upper bound** with the USL at $\mu + 2\sigma$. This means, $100\% - 2.5\% = 97.5\%$ of the deliveries are not to late.



(a) Example 1 (defect tolerance)

(b) Example 2 (delivery times)

Figure 2.12: Examples for 68-95-99.7-rule

Furthermore, the rule also introduces the term of **six sigma** or also lean six sigma. It basically means that processes operating with "Six Sigma quality" are assumed to have < 3.4 defects per million instances, so $\Pr(\text{error}) = 0.0000034$. It characterizes a process improvement approach. This likelihood is a combination of $\pm 6\sigma$ tolerance and a "drift" of $\pm 1.5\sigma$.

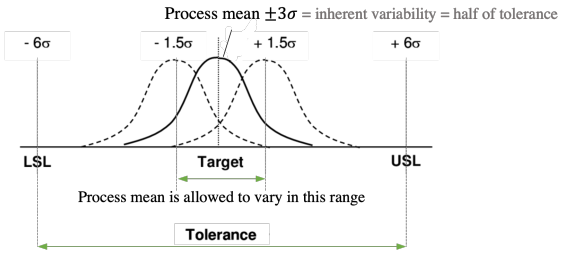


Figure 2.13: Lean six sigma

2.3 Data quality

In the introduction, we already looked at some key challenges regarding data quality. In this subsection, we will investigate and search for solutions for some of the following typical data quality problems in detail:

- **Incompleteness** - missing instances or attributes
- **Invalidity** - impossible values

- **Inconsitency** - conflicting values
- **Imprecision** - approximates or rounded values
- **Outdated** - values based on old observations

For that, we will take a look at missing, invalid, unlikely, and outlier values.

Missing values

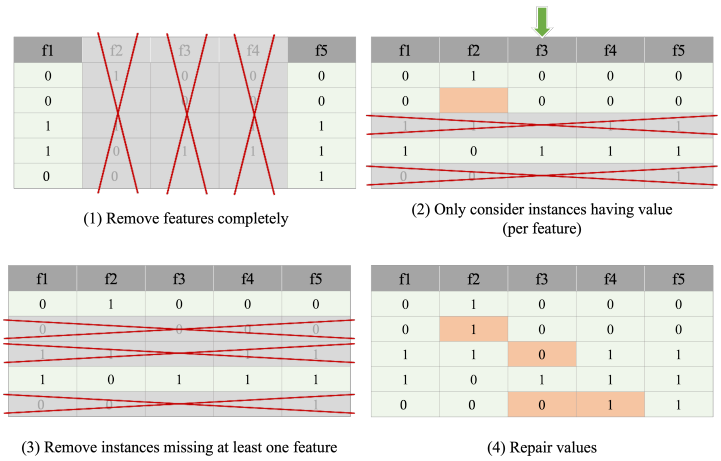
Imagine different missing features of some instances. Since some data is missing, we need to deal with this in some way. Here are the possible options: Missing features

1. Remove feature completely (for all instances)
2. Only consider instances that have a value (this is done for per-feature-evaluation)
3. Remove all instances that have one of the features missing
4. Repair missing features (imputation)

The problem setting and the possible solutions are visualized in 2.14.

f1	f2	f3	f4	f5
0	1	0	0	0
0		0	0	0
1	1		1	1
1	0	1	1	1
0	0			1

(a) Problem setting



(b) Possible solutions

Figure 2.14: Missing values

Impossible values

The next typical challenge is impossible values that by some mistake were entered as data. Examples are: Impossible values

- Wrong date format: instead of 2018-10-18, we would have 18-10-2018

- Completely impossible date or time: 2018-13-51, 23:61
- There can be spelling errors, for example for colors: Bllue
- The data type might not make sense with the feature, like the number of members as a float: 6.5 members

The handling of this problem is solved by either manually correcting entries or handling them just as for missing features.

Unlikely values

In contrast to impossible values, unlikely values are theoretically possible, but just not common to appear. Examples are:

- Age: 123 is rather unlikely, but possible
- Price: 120.000$ in a store where the other prices lie in the range of 5$ to 150$
- Dates: even on dates, where one would usually expect a uniform distribution over months and days, days 1 to 12 are more frequent than days 13 to 31[3]

Unlikely values, domain knowledge

Whether a value is unlikely or not is identified based on **domain knowledge**. They can then be further investigated to see, whether the unlikely value is actually valid.

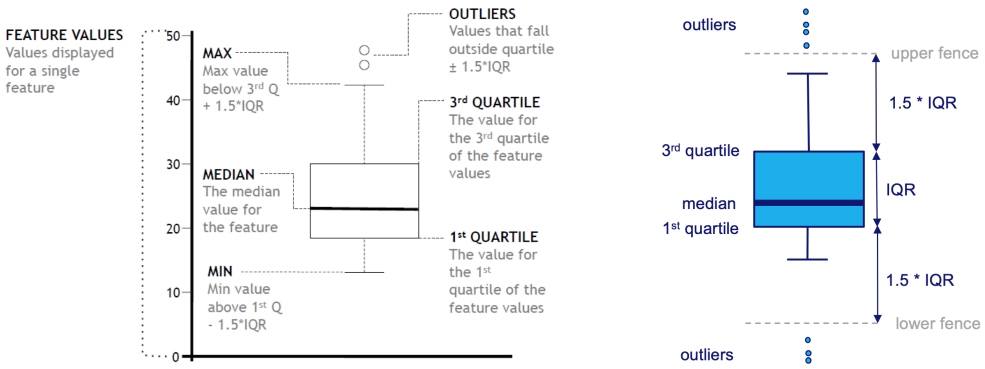
Outlier values

Outlier values

In contrast to unlikely values, **outlier values** are identified based on the distribution.

Box plot

An especially popular technique to visualize distributions and outliers is **box plots**. They were first introduced by John Tukey in the book "Exploratory data analysis" in 1977. Figure 2.15 shows the properties visualized by a box plot and also how to construct one given a data set.



(a) Properties on a box plot

(b) Construction of a box plot

Figure 2.15: Box plot

Let's first take a look at the properties one can see in the box diagram.

- The **median** value is depicted by the "Bar" in the center.

[3]NOT the case anymore if date format DD-MM-YYYY and MM-DD-YYYY are mixed

  - The median is the "middle" value, so the number is halfway between the lowest and highest number.
- The **IQR**, so the interquartile range, covering 50% of the "middle instances" is depicted by the "Box". IQR
  - The first quartile is the number halfway between the lowest and middle number, whereas the third is halfway between the middle and highest number.
  - The IQR is the distance between the first and third quartile.
- The upper whisker indicates the **maximal** value below the $3^{\text{rd}}\text{quartile} + 1.5 \cdot IQR$, whereas
- The lower whisker indicates the **minimal** value above the $1^{\text{st}}\text{quartile} - 1.5 \cdot IQR$.
- Finally, the **outliers** are drawn separately.

The description already contained a bit of the construction details, which will now be explained in more detail with an example. Consider the (already ordered) data set:

$\{_{1:}\ 1,\ _{2:}\ 2,\ _{3:}\ 5,\ _{4:}\ 7,\ _{5:}\ 8,\ _{6:}\ 8,\ _{7:}\ 9,\ _{8:}\ 9,\ _{9:}\ 9,\ _{10:}\ 10,\ _{11:}\ 10,\ _{12:}\ 10,\ _{13:}\ 11,\ _{14:}\ 12,\ _{15:}\ 14,\ _{16:}\ 19,\ _{17:}\ 23\}$

Then we construct the box diagram like this:

- The median value is 9 (at position 9).
- The first quatile has the value 8 (at position 5), the third one has the value 11 (at position 13) resulting in an $IQR = 11 - 8 = 3$.
- This means we have an upper fence $11 + 1.5 \cdot 3 = 15.5$, and the upper whisker as the maximum value below this fence at 14 (position 15).
- The lower fence has the value $8 - 1.5 \cdot 3 = 3.5$, the lower whisker therefore the minimum value above this fence value at 5 (position 3).
- Finally, the outliers are $1, 2, 19, 23$ (position 1, 2, 16, 17).

Those are all the necessary components to construct the box diagram.

Now, one final detail about box diagrams and also the topic of this paragraph is the handling of the outliers. They can first be removed (meaning remove values above and below the upper and lower fences), and their existence can be indicated by claiming the removed values to these thresholds. The process is shown in 2.16.

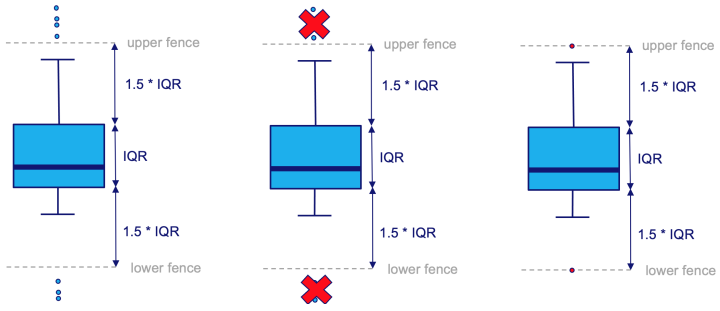


Figure 2.16: Handling outliers in box plots

2.4 Relations among features

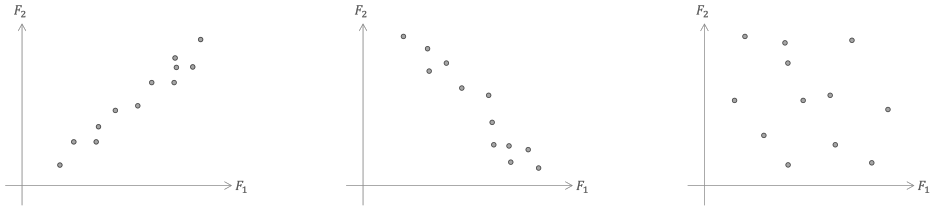
So far, we only really looked into features separately. This section will now focus on showing how multiple features are related. We will only consider relating TWO features,

but the techniques are also applicable to more.

Scatter plots

Correlation

As a first step to see whether a relation exists, first plot the two features. Examples for typical resulting **correlations** are shown in 2.17.



(a) Positive correlation (b) Negative correlation (c) No correlation

Figure 2.17: Scatter plots visualizing correlations

As an example for detecting correlations, we have a data set about basketball players with different features. The raw data will not be included here. Instead, we will directly look at all correlations of the features simultaneously. This is visualized using a **scatter plot matrix** (SPLOM) as in 2.18.

SPLOM

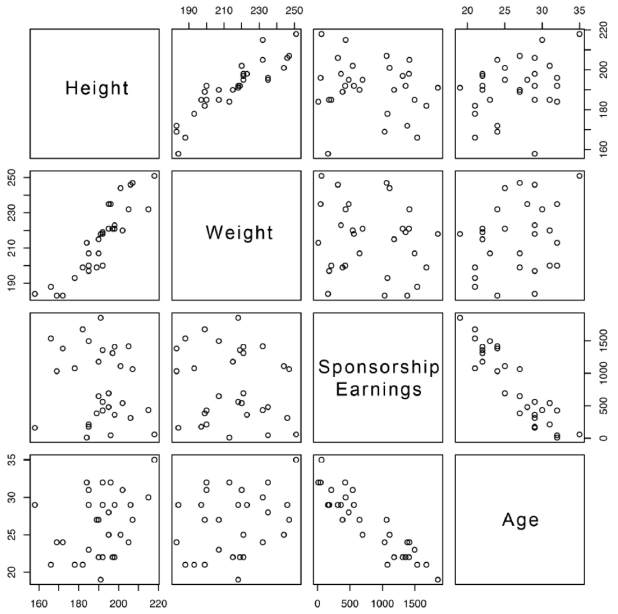


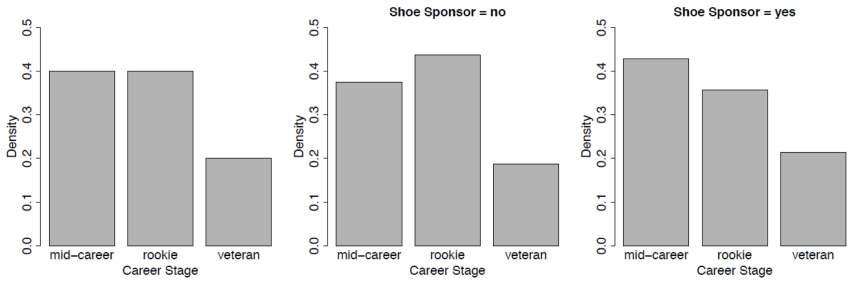
Figure 2.18: Scatter plot matrix for four features

Interesting to see in such a SPLOM is the mirror axis in opposite tiles of the matrix which is an absolutely linear line (so absolute correlation, or identity, as would be displayed if a feature would be displayed on a scatter plot compared to itself). This mirroring doesn't change the nature of a correlation. If feature A is positively correlated to feature B, the same goes in the other direction (equivalently for negative correlation).

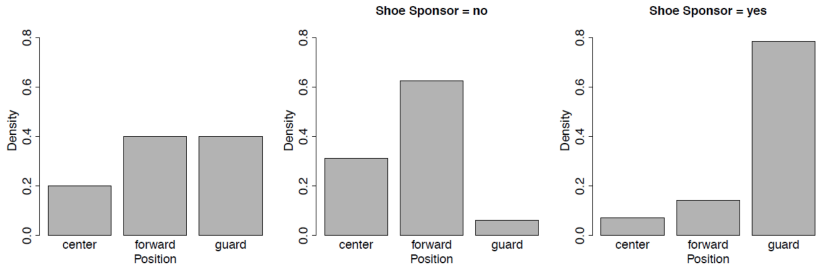
Collection of bar plots

Another way to display relations is via a collection of small multiple **bar plots**. Consider the plots in 2.19

Collection of small multiple bar plots



(a) Career stage: No relation to shoe sponsor



(b) Position: Strong relation to show sponsor

Figure 2.19: Collection of (conditioned) bar plots (Conditioned on shoe sponsor)

No relation is implied when the differently conditioned and non-conditioned bar plots don't show any significant difference, as for the example of career stages. On the other hand, a relation can be seen when the bar plots show specific differences. For example in the position case it can be seen that "guards" are more likely to have a shoe sponsor.

The difference and implied relation can be further highlighted by using **stacked bar plots** that show the conditioned percentage, as can be seen in 2.20.

Stacked bar plots

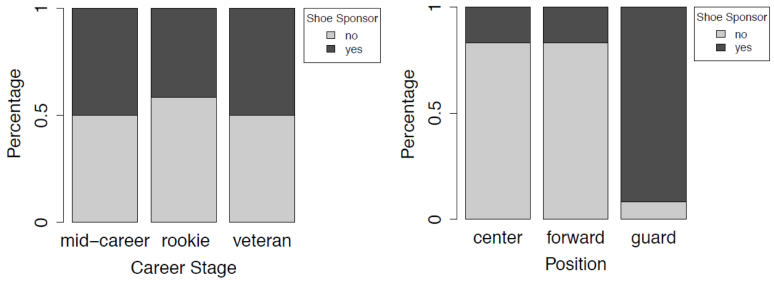
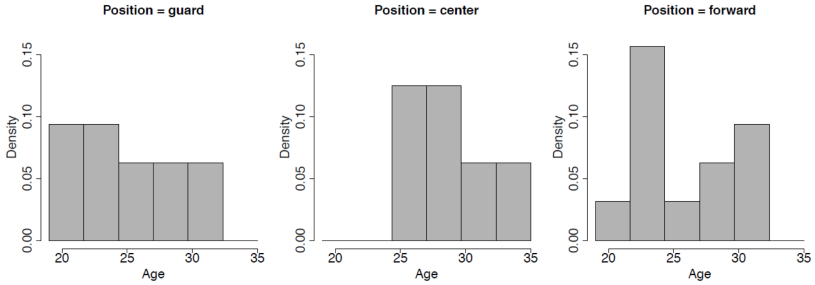


Figure 2.20: Stacked bar plots (for both career and position conditioned on shoe sponsor)

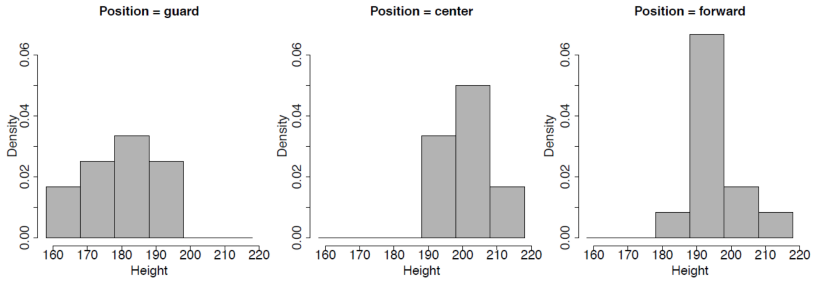
Collection of histograms and box plots

In the case of continuous variables, instead of bar plots, we can use a collection of small multiple **histograms**, as displayed in 2.21.

Collection of small multiple histograms



(a) Age (6 bins): No strong relation to position

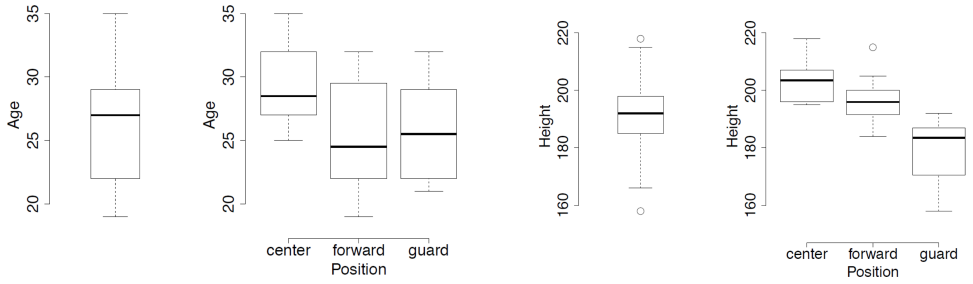


(b) Height (6 bins): Relation to position

Figure 2.21: Collection of (conditioned) histograms (Conditioned on position)

Collection of box plots

Alternatively, **box plots** can be collected and utilized to identify relations. Figure 2.22 conducts the same relation between age or height to position. It further highlights the relatively strict separation of age for individual positions.



(a) Age: Weaker relation to position

(b) Height: Stronger relation to position

Figure 2.22: Collection of (conditioned) box plots (Conditioned on position)

Descriptive statistics

To not only see the relation but also classify it with values, we now introduce some basic descriptive statistics. Based on n values $a_1, \dots, a_n$, we have the **sample mean** $\overline{a}$ and **sample variance** $var(a)$ and **standard deviation** $sd(a)$ as:

Sample mean

$$\overline{a} = \frac{1}{n} \sum_{i=1}^{n} a_i$$

Sample variance

$$var(a) = \frac{\sum_{i=1}^{n} (a_i - \overline{a})^2}{n-1}$$

$$sd(a) = \sqrt{var(a)} = \sqrt{\frac{\sum_{i=1}^{n}(a_i - \overline{a})^2}{n-1}}$$ Standard deviation

A quick note on why we divide by $n-1$ and not n for the calculation of $var(a)$. This is due to the estimated mean $\overline{a}$ instead of actual or true one $\hat{a}$. Simply put, since we can only estimate, we would rather overestimate the variance (divide by a smaller number) instead of underestimating (divide by a larger number) it. We would call a variance calculated by dividing by n a biased estimator.

To classify the relation between features, based on n pairs of values $(a_1, b_1), \ldots, (a_n, b_n)$ we have the **sample covariance** $cov(a, b)$ and the **correlation** $corr(a, b)$ as:

$$cov(a, b) = \frac{1}{n-1}\sum_{i=1}^{n}((a_i - \overline{a}) \times (b_i - \overline{b}))$$ Sample covariance

$$corr(a, b) = \frac{cov(a, b)}{sd(a) \times sd(b)}$$ Correlation

Covariance and correlation have the following properties:

$cov(a, b) \in [-\infty, \infty]$ (unbounded)

$cov(a, b)$ is positive $(+)$ / negative $(-)$ if a $(\pm)$ / $(\pm)$ and b $(\pm)$ / $(\mp)$ are the same / different

$corr(a, b) \in [-1, 1]$ (normalized)

$corr(a, b)$ > 0 positively correlated; < 0 if a and b are negatively correlated; $= 0$ independent

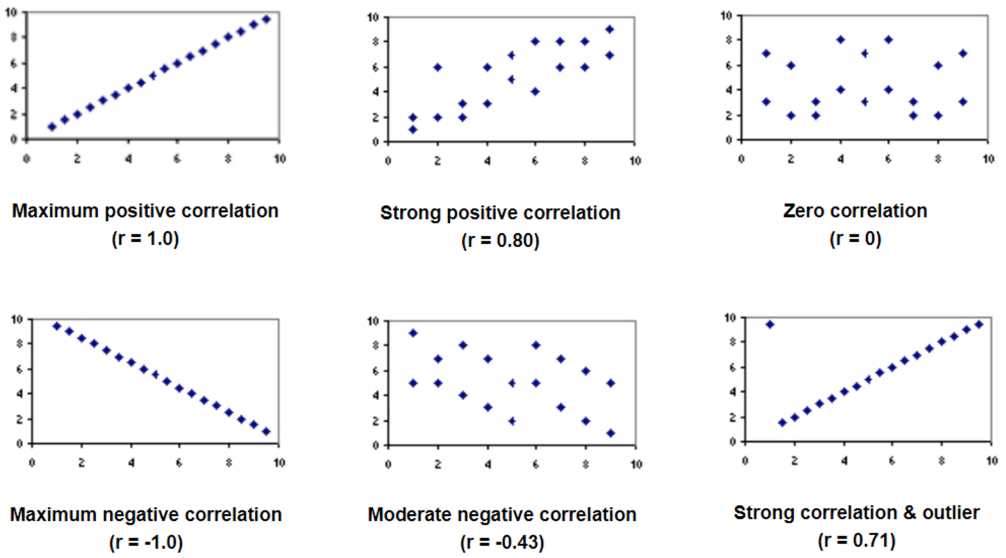


Figure 2.23: Different correlation examples with scatter plot

With the new knowledge, we can expand our previous SPLOM diagram with the **correlation matrix** values. A correlation matrix looks as follows:

Correlation matrix

$$corr\text{matrix}_{\{a,b,\ldots,z\}} = \begin{pmatrix} corr(a,a) & corr(a,b) & \cdots & corr(a,z) \\ corr(b,a) & corr(b,b) & \cdots & corr(b,z) \\ \vdots & \vdots & \ddots & \vdots \\ corr(z,a) & corr(z,b) & \cdots & corr(z,z) \end{pmatrix}$$

with $corr(f_1, f_2) = corr(f_2, f_1)$ for all $f_1, f_2 \in \{a, b, \ldots, z\}$ making the correlation matrix symmetric.

The updated SPLOM diagram, which basically contains the same information twice (just flipped) now also shows the correlation values as can be seen in 2.24.

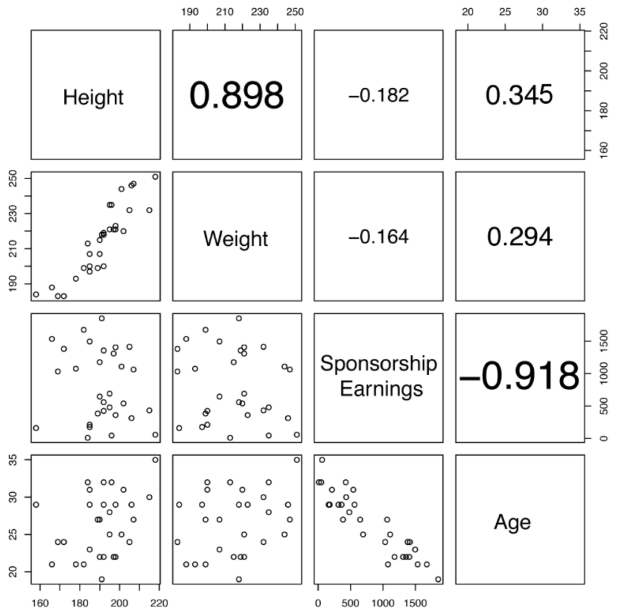


Figure 2.24: Scatter plot matrix of four features with according *corr*-values

2.5 Preparation for analysis

We now know some techniques to evaluate raw data, feature by feature and the relationship of features. But a very important step before applying analysis is the preparation, consisting of:

- **Normalization** to make things comparable,
- **Binning** to make things categorical, and
- **Sampling** to make data smaller or to change the bias.

Normalization

Normalization

Let's first start with normalization. Typically, when applying normalization, values are **mapped onto a predefined range** while maintaining differences. This predefined

range is usually something like $[0, 1]$ or $[-1, 1]$.

As an example, the following mapping transforms a_i into $a'_i \in [l, h]$ where

- l is the lower bound, h the upper one, and
- We have a complete set of values a with a defined minimum and maximum given.

$$a'_i = \frac{a_i - \min(a)}{\max(a) - \min(a)} \times (h - l) + l$$

Furthermore, we have the **standard score** using the standard deviation to normalize.

$$a'_i = \frac{a_i - \overline{a}}{sd(a)} \in [-\infty, \infty]$$

Standard score

Binning

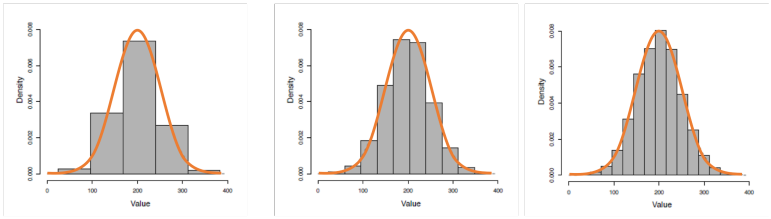
Next, we have binning of values to **turn continuous** features **into categorical** ones. Bins are a series of ranges. There are two approaches to determine the bins, both visualized in 2.25.

Binning

- **Equal-width binning** assumes a fixed width for all bins, where the number of items per bin may vary greatly.
- **Equal-frequency binning** on the other hand assumes a fixed number of items per bin, but allows a variable width.

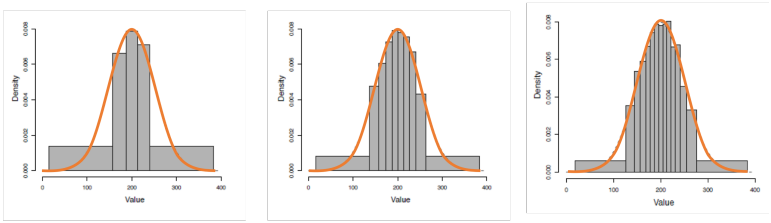
Equal-width binning

Equal-frequency binning



(a) 5 bins (b) 10 bins (c) 15 bins

(a) Equal-width bins



(a) 5 bins (b) 10 bins (c) 15 bins

(b) Equal-frequency bins

Figure 2.25: Binning with different number of bins

The number of items in a bin is reflected by its surface (need to consider both width and height).

Generally, the number of bins is important as we saw earlier with the problem of over-and under-fitting.

- Underfitting happens when the number of bins is too small, and information gets lost.
- Overfitting on the other hand occurs when the number of bins is too large, leading to sparseness with some (nearly) empty bins in areas where the true function considers items to occur very likely.

Sampling

Sampling

Finally, we have the preparation step of sampling (which actually ususally comes as the first step). It selects a subset of all available data, and thereby reduces the amount of data. Sampling can remove or introduce a **sample bias**.

There are different **types** of sampling, as depicted in figure 2.26.

(a) Top sampling

(b) Random sampling

(c) Stratified sampling

(d) Under-sampling

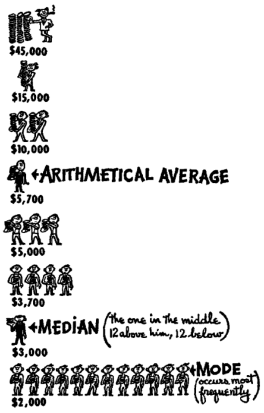
(e) Over-sampling (The green instances are replicas)

Figure 2.26: Types of sampling

- **Top** sampling takes the first n instances
- **Random** sampling
- **Stratified** sampling where the relative frequencies are ensured to be maintained (e.g. by taking the same percentage from every group)
- **Under**-sampling, where balance between groups is ensured by leaving out instances of over-represented groups.
- **Over**-sampling, where balance between groups is ensured by duplicating under-represented groups.

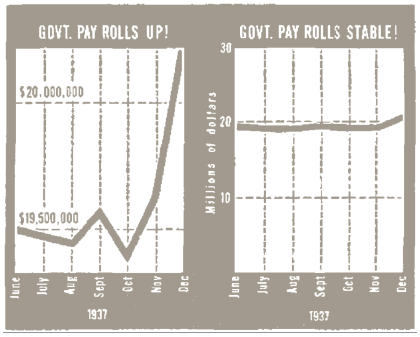
Good and poor visualizations

With statistics and especially visualization, one can really manipulate the view on the provided data, so basically introduces a very strong bias. There's actually a whole book about it, called "How to Lie with Statistics". The following images in 2.27 show some examples.



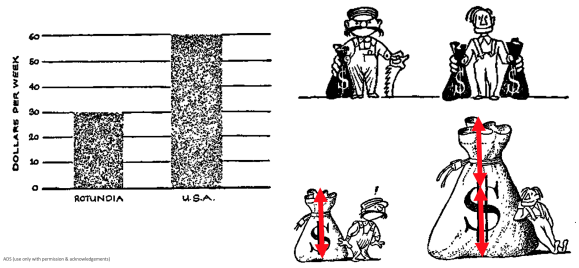
(a) Choosing the **middle**

- Arithmetical average (gets pumped up by rare but high maximum)
- Median (standing in middle)
- Mode (most frequently occurring)



(b) Tampering with **scales**

- Easy manipulation of difference perception



(c) Visualizing **factor** two

- Adjusting the whole volume according to height makes the difference seem more severe than just taking two instances

Figure 2.27: Manipulation of perception via statistics and visualization

As a good example, one can consider the information visualization book "The Visual Display of Quantitative Information" from Edward Rolf Tufte. It encourages the use of **data-rich illustrations** representing all available data such that:

- One can check individual values, while still
- Seeing trends and patterns when looking at the whole.

3 Decision trees

3.1 Statistics versus DM/ML

Statistics have been around for a while. Famous statisticians are for example:

- John Graunt (1620-1674), who studied London's death records around 1660.
  - He was able to predict the life expectance of a person at a particular age and was the first to create a "life table" with the probability of death for each age.
- Francis Galton (1822-1911), who introduced many core statistical concepts at the end of the 19th century.
  - He (re)invented variance, normal distribution, correlation, linear regression, etc.

Back then, statistics were concerned with the problem of making generalizations based on relatively little data. Since then, the availability of data changed drastically, with now having more of an overload of data. Therefore, more **pragmatic** instead of statistical approaches for handling large amounts of data were introduced to fuel the progress in data science.

- Major breakthroughs in the discovery of patterns and relationships are for example efficiently learning decision trees and association rules.
- By traditional statisticians, these were described as "data fishing", "data snooping", or "data degrading" (Surprisingly, some statisticians claim "owning" the data science field)

Modern statisticians are now also concerned with a more pragmatic approach. Leo Breiman (1928-2005) wrote a paper ("Statistical Modeling: The Two Cultures") about the two main camps of statisticians:

- The "classical statistics camp" (98%) assumes nature's behavior to fit some model and focuses on parameter estimation and goodness-of-fit tests.
  - An important aspect of this approach is **hypothesis testing**, which has led to the image of statisticians aiming to prove that nothing can be concluded from basically any given data while still data can be "tortured until confession", creating wrong conclusions.
- The other 2% of statisticians focus on simply finding a predictive function evaluated by predictive accuracy only (which fits the pragmatic approach).
- John W. Tukey (1915-2000), whose one of those 2% focussed on practical statistics.
  - This includes **exploratory data analysis** instead of hypothesis testing.
  - For example, he invented boxplots.

So to summarize the concept shift and also the difference between classical statistics and machine learning approaches:

- Before, a small amount of data or only samples of the whole data distribution were available, whereas
- Now, we have a big amount of data or all available data (due to computing power, storage, and tools).
  - The new approach is, therefore, to "let the data speak", since it's there.

Problems, even with this new approach, are:

- Data is always dirty, biased, etc. Fortunately, summarizing it can be surprisingly useful.
- Typical risks, raising the necessity of handling the new approach with care, are:
  - Testing of many hypotheses,
  - Over- or underfitting the data, and
  - Having a bias in the data or the representation.

3.2 Basics of decision trees

A typical decision tree looks as in image 3.1.

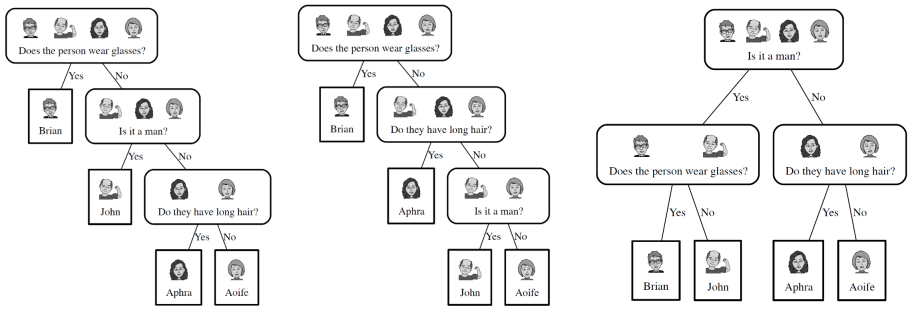


Figure 3.1: Decision tree for person distinction (different grouping)

The example shows that a **decision tree** is built by **grouping** instances step by step. In general, instances are partitioned into **increasingly smaller groups**. Decision tree

- How the groups are formed decides the outcome of the concrete decision tree (different trees are possible).
- For the grouping, keep two goals in mind:
  1. The tree shall be as small and simple as possible.
  2. The leaves shall be homogeneous in terms of the target feature.

The overall **goal of a decision tree** is to explain the target feature in terms of the descriptive features, so we have a supervised learning scenario.

- For categorical features we can differentiate based on the different classes.
- For numerical features, we need to define a threshold or something similar, to make a decision.

The following example in 3.2 (life expectancy given different features) shows the derivation of a (more or less) valid decision tree given tabular example data.

So summarized, a decision tree consists of three different types of nodes: Decision tree components

- A **root node** referring to all instances,
- **Interior nodes** partitioning the set of instances based on a descriptive feature, and
- **Leaf nodes** that have a label (the target feature value) that hopefully corresponds to a homogeneous group of instances with the same label.

How the partitioning influences the size and therefore efficiency of the decision tree can

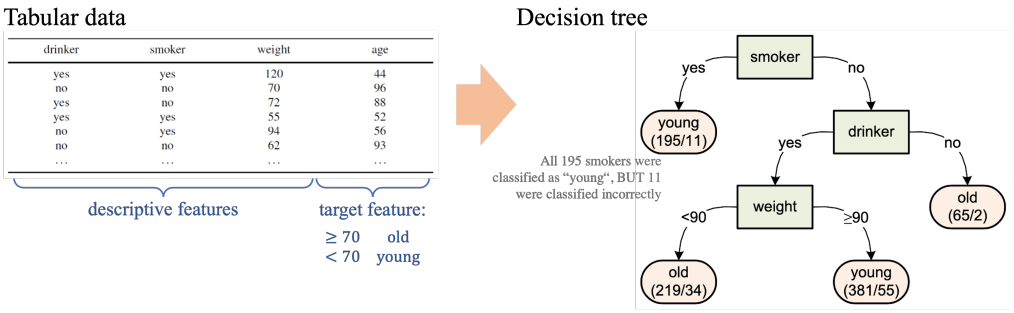


Figure 3.2: Example for deriving a decision tree from tabular data (life expectancy)

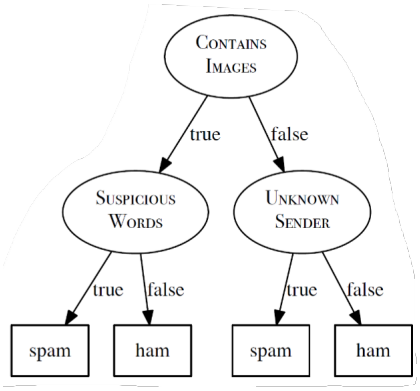
be seen in the example in 3.3. Both the good and bad partitioning options classify the observed instances correctly, but one is more simple and seems better. While investigating the example, keep the following keywords in mind:

Partitioning keywords

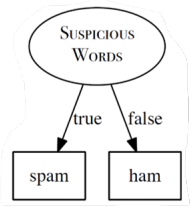
- Avoid overfitting
- Apply Occam's razor (problem-solving principle recommending searching for explanation constructed with the smallest possible set of elements = simplest solution is best one)
- Prefer shallow trees

ID	Suspicious Words	Unknown Sender	Contains Images	Class
376	true	false	true	spam
489	true	true	false	spam
541	true	true	false	spam
693	false	true	true	ham
782	false	false	false	ham
976	false	false	false	ham

(a) Tabular data



(b) "Bad" partitioning



(c) "Good" partitioning (fewer decisions)

Figure 3.3: Example for different partitioning results on the same problem (both correct)

3.3 Entropy

As a main motivation of why we need the term entropy, let's first look at the idea of **information gain**. This can be applied to decision trees and asks for improvement

Information gain

in knowledge with each partitioning step, so better predictability of class labels in the nodes. This implies more homogenous interior nodes with every layer as visualized in the example in 3.4.

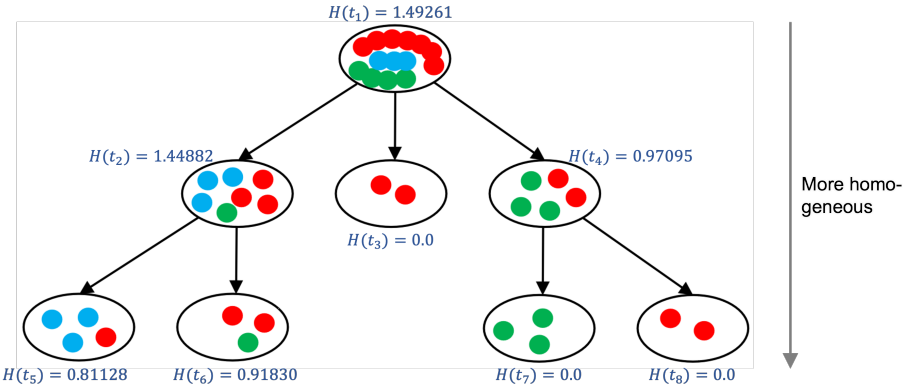


Figure 3.4: Idea of information gain and entropy intuition

Next, we'll take a look into the intuition of the term **entropy**. Figure 3.4 also displays the entropy values. As one can see in the example: Entropy

- Entropy **measures the impurity** in a set.
- With higher entropy, the **uncertainty in guessing** a class label grows.
- For a low entropy, the information gained when investigating the according data set is not very high, basically the data is "compressable". Entropy therefore also indicates **incompressibility**.
- Or put alternatively: entropy represents the number of bits needed to encode one instance knowing the population it comes from.

All of these statements are summarized in the formula:

$$H(t) = -\sum_{i=1}^{n} \left(\Pr[t = i] \cdot \log_s(\Pr[t = i]) \right)$$

The minus occurs, since $log_s(\frac{1}{x}) = -\log_s(x)$. In this course, we will always take the logarithmic base $s = 2$.

For a better understanding, we will calculate the entropy for three example sets from figure 3.4.

Example	Distribution over colored dots
1: high entropy value	$n_{\text{red}} = 7, n_{\text{blue}} = 3, n_{\text{green}} = 4$, so $n = 14$
$\Longrightarrow H(t_1) = -\left(\frac{7}{14} \cdot \log_2 \left(\frac{7}{14}\right) + \frac{3}{14} \cdot \log_2 \left(\frac{3}{14}\right) + \frac{4}{14} \cdot \log_2 \left(\frac{4}{14}\right) \right) = 1.49261$	
2: middle-high entropy value	$n_{\text{red}} = 2, n_{\text{blue}} = 0, n_{\text{green}} = 3$, so $n = 5$
$\Longrightarrow H(t_4) = -\left(\frac{2}{5} \cdot \log_2 \left(\frac{2}{5}\right) + \frac{2}{5} \cdot \log_2 \left(\frac{2}{5}\right) \right) = 0.97095$	
3: minimal entropy value	$n_{\text{red}} = 0, n_{\text{blue}} = 0, n_{\text{green}} = 3$, so $n = 3$
$\Longrightarrow H(t_7) = -\left(\frac{3}{3} \cdot \log_2 \left(\frac{3}{3}\right) \right) = 0$	

Now that we have seen an example, we can easily see the **bounds of entropies**. Bounds on H

- The lowest possible entropy value yields when all instances have the same value, then $H(t) = 0$.
  - Then there is no impurity at all, no uncertainty when guessing, and the information in the data is highly compressible.
- The highest possible entropy value yields when we have an even distribution over all possible values, then $H(t) = -n\left(\frac{1}{n} \cdot \log_2\left(\frac{1}{n}\right)\right) = \log_2(n)$ is maximized
  - E.g., for 3 possible values: $log_2(3) \approx 1.58$
  - Then there is the highest possible impurity, highest uncertainty when guessing, and the information in the data is incompressible.

Our goal when building decision trees is to have **pure leaves**, or the lowest possible average over the entropies of all leaves. With the entropy, we can now also put a number to the concept of information gain or loss, as can be seen in 3.5. When we have to select the next decision dividing an interior node, we select the features partitioning into groups with the least **remaining entropy** rem, which is the weighted average over all subnodes.

rem

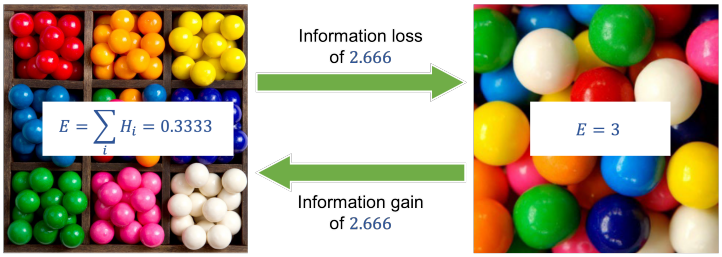


Figure 3.5: Example for information gain and loss

3.4 ID3 algorithm

We now know about the core concepts and components for building decision trees. The following algorithm introduces a standard procedure to build decision trees.

ID3, short for **iterative Dicchotomiser 3**, was developed by Ross Quinlan in 1986 and is a predecessor of algorithms like C4.5. The key idea is the following:

ID3

1. Calculate the entropy of every attribute using the data set D.
2. Split the set D into subsets using the attribute with the minimal resulting entropy.
   - Meant is the entropy after splitting using this attribute.
   - Equivalent formulation: choose the attribute maximizing the information gain.
3. Make a decision tree node containing that attribute.
4. Recurse on the subsets using the remaining attributes.

More formally, the algorithm looks as follows in pseudocode. We will afterward investigate certain details of the algorithm 3.1.

- Detail 1: three different reasons to stop
  - All instances have the same classification (labeled with consensus value)
  - No features are left (labeled with majority value)
  - Data set is empty (labeled with the majority value of the parent node)

```
Require: set of descriptive features d
Require: set of training instances 𝒟

// Different reasons to stop
if all instances in 𝒟 have the same target level C then
  return DecisionTree(leaf_node with label C)
else if d is empty then
  return DecisionTree(leaf_node with the label of majority target level in 𝒟)
else if 𝒟 is empty then
  return DecisionTree(leaf_node with label of the majority target level of the immediate parent
    node)

else
  // Pick feature
  d[best] ← arg max_{d∈d} IG(d, 𝒟)
  make a new node Node_{d[best]} and label it with d[best]
  partition 𝒟 using d[best]
  remove d[best] from d

  // Create subproblems
  for each partition 𝒟_i of 𝒟 do
    grow a branch from Node_{d[best]} to the decision tree created by rerunning ID3 with 𝒟 = 𝒟_i
```

Listing 3.1: ID3 algorithm code

- Detail 2: which feature to select
  - The feature should be picked that maximizes the information gain.
  - A feature can't be picked more than once along the path from the root.
- Detail 3: subproblems are created based on the selected feature, building the decision tree in a divide-and-conquer fashion.
  - The amount of subproblems is dependent on the selected feature.

The ID3 algorithm has a lot of variations, from which we're gonna take a look at some.

Alternative information gain notions

Information gain (IG) aims to measure the improvement in purity, predictability, and compressibility. Instead, it is also possible to select a feature maximizing:

- The information gain ratio (GR), or
- The Gini index ($Gini$)

The standard information gain notion favors features with many values since a split in many subsets increases the entropy. The **information gain ratio** on the other hand addresses:

Information gain ratio GR

$$GR(d, \mathcal{D}) = \frac{IG(d, \mathcal{D})}{\underbrace{-\sum_{l \in levels(d)} \Pr[d = l] \cdot \log_2(\Pr[d = l])}_{\text{Basically: make absolute value relative}}}$$

Another alternative is the **Gini index** measuring impurity in an alternative way. Specifically, it uses the expected misclassification rate when guessing based on the observed

Gini index $Gini$

distribution:

$$Gini(t, \mathcal{D}) = 1 - \underbrace{\sum_{l \in levels(t)} \underbrace{\Pr[t = l]^2}_{\substack{\text{Guess } t=l \text{ with probability } \Pr[t=l] \text{ and} \\ \text{with probability } \Pr[t=l] \text{ this is right}}}}_{\text{Probability of guessing the wrong label}}$$

A comparison of all the evaluation metrics can be found in figure 3.6.

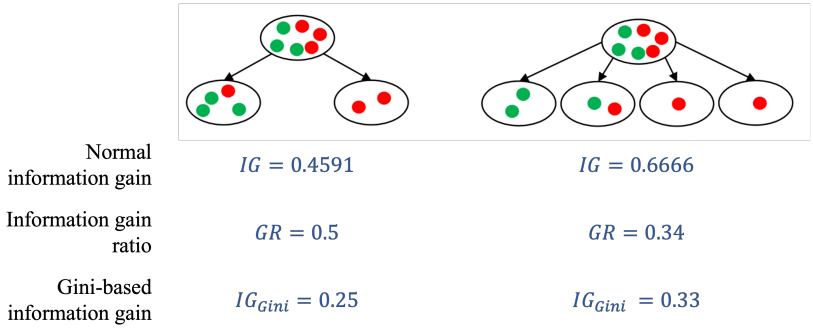


Figure 3.6: Comparison of IG, GR, and $Gini$

Pruning decision trees

When applying the ID3-algorithm to build decision trees, two possible problems can occur: The decision tree overfits the data, or is too complex or deep. For those problems, there are now two solution directions:

- Pre-pruning, so an early stopping, functioning forwards, and
- Post-pruning, so a reduced error, functioning backward.

Pre-pruning

We'll first take a deeper look into **pre-pruning**, where at some point the procedure of creating subtrees is **stopped** at which point the label is determined via majority vote. There are many possible stopping criteria, for example, lower bounds on the number of instances or on the information gain.

The trees resulting from pre-pruning may not be consistent with respect to the data. But they generalize and therefore avoid overfitting nicely. So the procedure is very efficient, but strong dependencies at lower levels of the tree might be missed.

Post-pruning

Next, let's investigate **post-pruning**, where first the whole decision tree is built and then some branches are **cut-off** that don't add too much information. One common approach for cutting off is to

- First split the data in a **training set** and a **validation/test set**,
- Built the decision tree based on the training set, and then
- Measure the performance of each split based on the validation/test set (e.g., count misclassified instances).

Ensembles

The final extension works with **ensembles**, where: Ensembles

- Instead of creating a single decision tree, a set of trees called a "model ensemble" is created.
- The models should complement each other.
- Different models can then "vote" on a label (votes may be weighted).

The concept relates to the "wisdom of the crowds" and aims at avoiding overfitting. The multiple trees may give different answers. Multiple trees can give different answers, then a "compromise" should be found, so the most frequent value or average is selected. There are many variations of this idea.

The first implementation of ensembles is called **boosting** where an iterative correction in a sequential manner happens. Boosting

- The correction happens by changing the data set based on previous misclassifications.
- Wrongly classified instances get a higher weight for training the next model.
- This creates a sequence of models, that combined lead to a (rather) correct classification.

The second implementation is called **bagging** where data is split upfront allowing a parallel rather than sequential model training. We alter the original data set by adjusting the rows (so the instances). Bagging

- Each model is based on a random sample of the data set.
- The idea is, that models are avoided depending on a specific sample in the data set. This is supposed to avoid that learning in decision trees is very sensitive to small variations.
- How the training bags are created has a lot of variants. Instances can for example be removed, or other duplicated, etc.

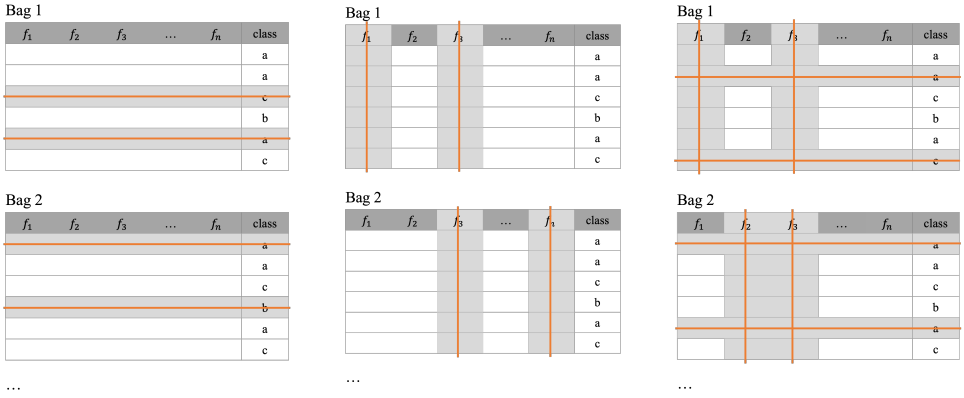
A third implementation called **subspace sampling** takes a similar approach as bagging, but instead of altering the rows, the columns so the features are adjusted. Subspace sampling

- Each model is based on a random set of descriptive features.
- The idea is that overfitting is less likely and also the training process is faster when focussed on just a few features instead of the whole feature space.
- The similarity to (instance) bagging is highlighted with the alternative name "feature bagging".

A fourth implementation combines feature and instance bagging and is called **random forest**. All three bagging strategies are also visualized in 3.7. Random forest

Finally, all of the ensemble methods can be combined in many possible combinations.

- When we have multiple classifiers (e.g. implemented as decision trees), they can be combined by voting or averaging.
- This often leads to a higher accuracy on unseen data and avoids overfitting.



(a) (Instance) bagging

(b) Subspace sampling
(Also called feature bagging)

(c) Random forest

Figure 3.7: Different ensemble bagging implementations

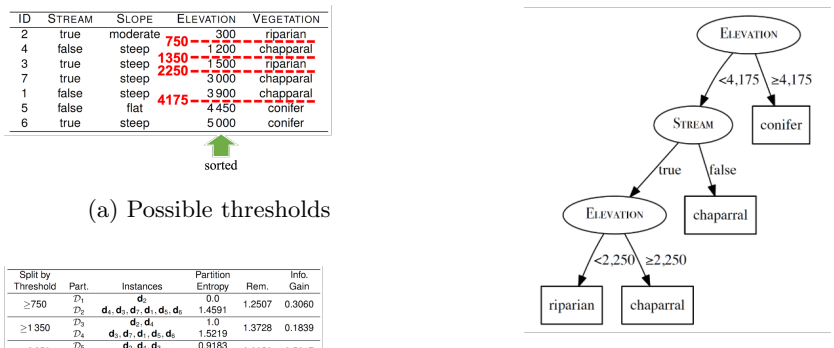
3.5 Dealing with continuous variables

So far we mainly assumed categorical features, for both the descriptive and target features. Now, we also want to take a look at how to deal with continuous features. Remember, that generally continuous features can be tuned into categorical ones using binning.

First, let's look at **continuous descriptive features**. The challenge is to determine suitable boundaries, where also an infinite number of thresholds is possible. The approach is:

- Sort the instances based on the continuous feature and then look for changes in class labels.
- Points where class label changes happen are candidate thresholds.
- The threshold with the highest information gain is selected.

How this works is visualized as an example in 3.8.



(a) Possible thresholds

(b) Thresholds selection

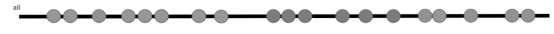
(c) Final tree

Note: the same feature can be used twice along a path.

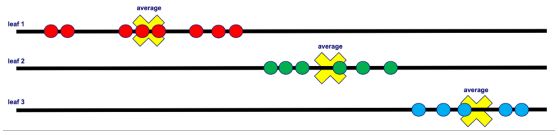
Figure 3.8: Building a decision tree with a continuous descriptive feature

Next, consider a **continuous target feature**. Here, we want to find descriptive features that "nicely" partition the target feature axis.

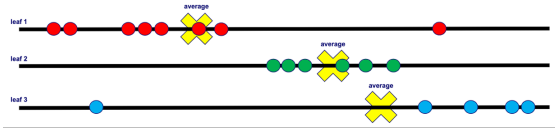
- How to distinguish and evaluate different classifications can be seen in 3.9
- Impurity is measured by the variance within a partition (so a leaf in the decision tree)
- But the split can't use the target feature as a decision. Instead, the weighted variance after the split is used as a performance criterion
  - The smaller, the better.
  - $var(t, \mathcal{D}) = \frac{1}{n-1} \sum_{i=1}^{n} (t_i - \bar{t})^2$



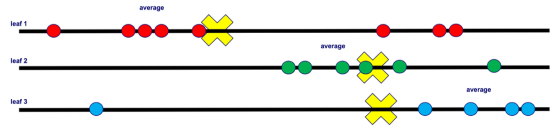
(a) Original dataset



(b) Good classification



(c) Reasonable classification



(d) Poor classification

Figure 3.9: Level of validity of different classifications for a continuous target feature

It's important to keep track of over- and underfitting, as visualized in 3.10. Generally, variance gets smaller when the sets get smaller, so the measure leans towards overfitting. This can be avoided by stopping early enough.

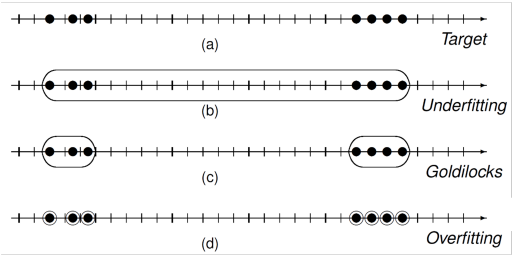


Figure 3.10: Over- and underfitting in continuous target feature classification

The **predicted value**, so the output of the decision tree, is the **average** value within a (leaf) node.

To be able to still use the ID3-algorithm, the decision for $\mathbf{d}[best]$ needs to be adjusted.

- Instead of $\mathbf{d}[best] \leftarrow \arg\max_{d \in \mathbf{d}} IG(d, \mathcal{D})$, we will
- Base the split on the feature lowering the weighted variance within the subtrees as much as possible, so $\mathbf{b}[best] \leftarrow \arg\min_{d \in \mathbf{d}} \sum_{l \in levels(d)} \frac{|\mathcal{D}_{d=l}|}{|\mathcal{D}|} \cdot var(t, \mathcal{D}_{d=;})$

In general, there are some extensions of ID3 implementing different ideas presented throughout this chapter:

- ID3 is the original.
- C4.5 can also deal with continuous features, missing values, implements post-pruning, etc. (C5.0 extends this further)
- J48 is an open-source implementation of C4.5.
- CART (classification and regression trees) use the Gini index, can handle continuous features, and can deal with missing values.
- CAID (Chi-square automatic interaction detector) relies on the Chi-square test to determine the best next split at each stop.

4 Regression

In the next three chapters, we'll look at **error-based learning**. In general, this learning approach functions as follows:

Error-based learning

- We have a **parameterized prediction model** which is initialized with random parameters.
- An **error function** is then used to evaluate the performance of this model when it makes predictions for instances in a training dataset.
- Based on the results of the error function, the parameters are **iteratively adjusted** to create a more and more accurate model.

Error function

There are different approaches to realizing error-based learning:

- Regression (covered in this section)
- SVMs (covered in the next section)
- Neural networks (covered in a later section)
- Genetic algorithms, or other evolutionary approaches

We'll start with **regression** and the following basic idea.

Regression

$$\begin{aligned} \text{Our model: } & f : \text{descriptive features} \to \text{target features} \\ \text{Goal: find } & f \text{ minimizing } error(\text{prediction}, \text{observed data}) \end{aligned}$$

When we compare this approach to decision trees, we see:

- Decision trees were initially developed for categorical features and then extended to continuous features.
- Regression followed the reverse path, which means it's most **suitable for continuous data**.
- Still, both are supervised learning techniques.

4.1 Simple linear regression

Consider the following simple example where we have:

- Rental price p_r as our target feature, and
- Size s as our descriptive (continuous) feature

We assume a linear dependency $p_r = b + a \cdot s$ and now want to base our prediction of the rental prize on the size. The example will guide us through this subchapter.

The **general problem** is given as follows:

General problem regression

- We have given n data rows in a set $\mathcal{D}$ with a target feature t and descriptive features $\mathbf{d} = (d[1], d[2], \cdots, d[m])$, and
- We want to find a regression function $\mathbb{M}_{\mathbf{w}}$ with a constant weight and a weight for each feature, where
- We predict $\text{pred}(t) = \mathbb{M}_{\mathbf{w}}(\mathbf{d}) = \mathbb{M}_{(w[0],w[1],\cdots,w[m])}(d[1], d[2], \cdots, d[m])$

In our example, we only have one descriptive feature $\mathbf{d} = (d[1])$, two weights $\mathbf{w} =$

$(w[0], w[1])$, and the regression function is linear, so $\underbrace{\mathbb{M}_{\mathbf{w}}(d)}_{p_r} = \underbrace{w[0]}_{b} + \underbrace{w[1]}_{a} \cdot \underbrace{d[1]}_{s}$.

What can be seen is that the weights $\mathbf{w} = (w[0], w[1])$ define the linear function. Our goal is now to find the weights such that the resulting function has the "smallest error". There are different ways to characterize the error with different **error metrics**:

Error metrics

Sum of squared errors L_2

- Sum of squared errors

$$L_2(\mathbb{M}_{\mathbf{w}}, \mathcal{D}) = \frac{1}{2} \cdot \sum_{i=1}^{n} (t_i - \mathbb{M}_{\mathbf{w}}(d_i))^2$$

  - For each instance: compute the error, then square it
  - Compute the sum of the results
  - Finally, take half ($\rightarrow$ end result)

Mean squared error MSE

- Mean squared error

$$MSE(\mathbb{M}_{\mathbf{w}}, \mathcal{D}) = \frac{1}{n} \cdot \sum_{i=1}^{n} t_i - \mathbb{M}_{\mathbf{w}}(d_i)$$

Root mean squared error $RMSE$

- Root mean squared error

$$RMSE(\mathbb{M}_{\mathbf{w}}, \mathcal{D}) = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} t_i - \mathbb{M}_{\mathbf{w}}(d_i)}$$

Mean absolute error MAE

- Mean absolute error

$$MAE(\mathbb{M}_{\mathbf{w}}, \mathcal{D}) = \frac{1}{n} \cdot \sum_{i=1}^{n} \mid t_i - \mathbb{M}_{\mathbf{w}}(d_i) \mid$$

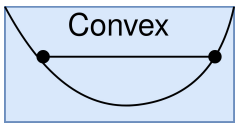
All of the introduced error metrics express the same idea, but usually, L_2 is chosen since it has a simple derivative. We'll take a look at the **partial derivative of error**, concretely of L_2 (here, i refers to the training instance, and j to one of the multiple descriptive features)

$$\begin{aligned} \frac{\delta}{\delta \mathbf{w}[j]} L_2(\mathbb{M}_{\mathbf{w}}, \mathcal{D}) &= \frac{\delta}{\delta \mathbf{w}[j]} \frac{1}{2} \sum_{i=1}^{n} (t_i - \mathbb{M}_{\mathbf{w}}(d_i))^2 \\ &= -\sum_{i=1}^{n} \left((t_i - \mathbb{M}_{\mathbf{w}}(d_i)) \cdot d_i[j] \right) \end{aligned}$$

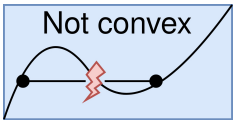
Using this derivate, we can now find the weights minimizing the error.

- Brute force, meaning we try as many values as possible for the weights, isn't feasible in practice (not even for simple linear case).
- But we can use that our error surface is convex and therefore has a global minimum, enabling faster methods.
  - Take the partial derivatives (for linear case: $\delta / \delta \mathbf{w}[0]\ L_2(\mathbb{M}_{\mathbf{w}}, \mathcal{D})$, and $\delta / \delta \mathbf{w}[1]\ L_2(\mathbb{M}_{\mathbf{w}}, \mathcal{D})$)
  - Find the correct values for all partial derivatives to result in zero (Needs to be the case for an actual global minimum)

One important thing to mention, the convex property of a function is very useful here:



- One local minimum = global minimum
- Gradient descent works



- Multiple local minima, so global minimum harder to find
- Gradient descent might fail

Figure 4.1: Convex vs. non-convex functions

The technique we can use to quickly find the global minimum is **gradient descent**. For convex functions we know, that we will always end up in the glocal minimum. Still, there is the open question of which direction to walk to: Gradient descent

- Take the steepest way down, which is known since there is a derivative of the function.
- This leads to a lower point point on each step and therefore converges.

An important decision to make is the **step size**: Step size

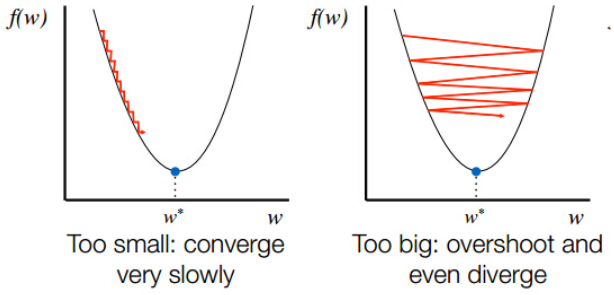


Figure 4.2: Step size for gradient descent

4.2 Multiple descriptive features

We now assume a feature consisting of multiple elements:

$$\begin{aligned}\mathbb{M}_{\mathbf{w}}(\mathbf{d}) &= \mathbf{w}[0] + \mathbf{w}[1]\mathbf{d}[1] + \cdots + \mathbf{w}[m]\mathbf{d}[m] \\ &= \mathbf{w}[0] + \sum_{j=1}^{m} \underbrace{\mathbf{w}[j]}_{\text{weight of the } i\text{-th feature } d[j]} \mathbf{d}[j] \\ &= \sum_{j=0}^{m} \mathbf{w}[j] \underbrace{\mathbf{d}[j]}_{\text{with } d[0]=1} \\ &= \underbrace{\mathbf{w} \cdot \mathbf{d}}_{\text{dot product of vectors}}\end{aligned}$$

In particular, this means we have $m+1$-dimensional vectors $\mathbf{d}_i$ and $\mathbf{w}$ with m as the number of features. This notational convenience extends the normal feature vector $\mathbf{d}'_i$

by $\mathbf{d}_i[0] = 1$. With n instances, we therefore have the error function:

$$L_2(\mathbb{M}_{\mathbf{w}}, \mathcal{D}) = \frac{1}{2}\sum_{i=1}^{n}(t_i - \underbrace{\mathbf{w} \cdot \mathbf{d}_i}_{=\mathbb{M}_{\mathbf{w}}(\mathbf{d}_i)})^2$$

With our now introduced multiple descriptive feature vector, we can write down the **sketch of** the overall **algorithm** of regression:

```
1  Require: set of descriptive features D
2  Require: learning rate α (controls how quickly algorithm converges)
3  Require: function Δ_error (determines the direction in which to adjust given weight w[i] to move
       down the slope of an error surface determined by D)
4  Require: convergence criterion (indicating when the algorithm has been completed)
5
6  w ← random starting point in weight space // randomly pick initial point
7  repeat
8    for each w[j] ∈ w do
9      // run downhill in steepest direction with speed α
10     w[j] ← w[j] + αΔ_error(D, w)[j]
11 until convergence occurs // stop when improvements become too small
```

Listing 4.1: Sketch of regression algorithm

For Δ_{error} we typically choose $\frac{\delta}{\delta \mathbf{w}[j]} L_2$. This means:

- If the derivative is positive, lower the weight $\mathbf{w}[j]$,
- If the derivative is negative, increase the weight $\mathbf{w}[j]$,
- While $\alpha > 0$ determines the speed in both cases.
- Line 10 in 4.1 would then result in $\mathbf{w}[j] \leftarrow \mathbf{w}[j] - \alpha \sum_{i=1}^{n}(t_i - \mathbf{w} \cdot \mathbf{d}_i)\mathbf{d}_i[j]$

4.3 Interpretation of results

For the one-feature case, the interpretation isn't difficult. We can simply see, that the target feature has some linear dependence on the descriptive feature. E.g., in our earlier example: the rental price has a "direct" dependency on the size.

It becomes more difficult when **interpreting results for multiple descriptive features**. This is due to the potentially completely different ranges of those features. The weights change dramatically when the units change (e.g. when changing cm^2 to m^2). This shows that the weight itself is irrelevant, only the sign has meaning.

Significance test

An alternative approach is the **significance test**.

In a simple version of this, we have:

- Create a regression model using k descriptive features and measure the error Δ'.
- Create k regression models leaving out one descriptive feature at a time and measure the error Δ_i for $i \in [k]$.
- The difference in error $|\Delta' - \Delta_i|$ indicates the significance of feature i for $i \in [k]$

In a more complex way (no need to know details), we follow this approach:

- Null hypothesis: the feature does *not* have a significant effect on the model.

- Null hypothesis is rejected when *p-level* is too small ($1 - 5\%$) $\rightarrow$ a smaller p-value indicates a more important feature.
- In statistical hypothesis testing:
  - p-value or probability value: the probability of obtaining test results at least as extreme as results actually obtained during testing
  - Assumes a correct null hypothesis
  - Very small p-value means: such an extreme observed outcome is very unlikely under the null hypothesis

4.4 Handling categorical features

So far we assumed features (both descriptive and target) are continuous. In this chapter, we'll look at categorical descriptive and target features.

- E.g., {true, false}, {A, B, C}, or similar

For **categorical descriptive features**, we'll introduce $\{0, 1\}$-features for every possible value, which is called **one hot encoding**.

One hot encoding

- E.g., for {A, B, C}: A= $(1, 0, 0)$, B= $(0, 1, 0)$, C= $(0, 0, 1)$

Next to one hot encoding, we have **single numeric value encoding**:

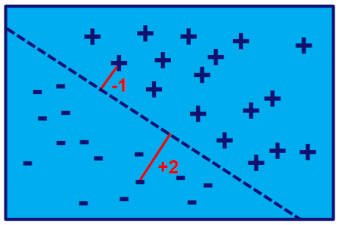
- Binary values {true, false} which can be translated to {0, 1}.
- Also, categorical variables with a clear order (ordinal) like {good, average, poor} can be translated to {1.0, 0.5, 0.0}.

Possible issues are:

- Adding order to unordered categorical variables resulting in nonsense
  - E.g., simple encoding applied to country names maps numbers to countries (but no natural order exists)
- All encodings are approximations, so intermediate values will be considered possible by the "regression machine"
- One hot encoding discards dependencies
  - Say if A= 1 then logically B= 0, but B= 0.66 is also possible
- Approach may introduce many additional features, making the problem computationally expensive

For a **categorical target feature** we can try to derive a numerical (continuous) feature. The naive approach is to find a line separating the results (try to find a line s.t. $\mathbf{wd} = 0$). The line then performs more of a separation than a prediction.

x	y	class
2.1	2.0	+
3.1	1.4	+
1.2	0.8	-
2.9	3.3	+
0.8	1.1	-
...	...	..



Target feature is +/- label, and not the value on the y-axis

Figure 4.3: Seperating target feature (alternative to prediction)

This means, we can use the $\{0, 1\}$ idea for encoding the two catgories (+/-). The model then also produces either 0 or 1:

$$\mathbb{M}_{\mathbf{w}}(\mathbf{d}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{d} \geq 0 \\ 0 & \text{else} \end{cases}$$

After this step, do business as usual e.g. by minimizing the sum of squared errors.

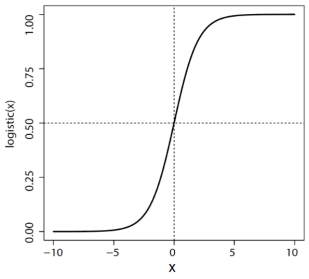
- Notice that in this naive approach, we don't use the distance to the decision boundary yet.
- But this would be desirable to make the decision surface continuous or smooth and thereby more applicable to gradient descent.

4.5 Logistic regression

Logistic regression will help us make a 0/1-decision continuous and smooth.

Logistic function

First, we need the **logisitic function**:



$$logistic(x) = \frac{1}{1 + e^{-x}}$$

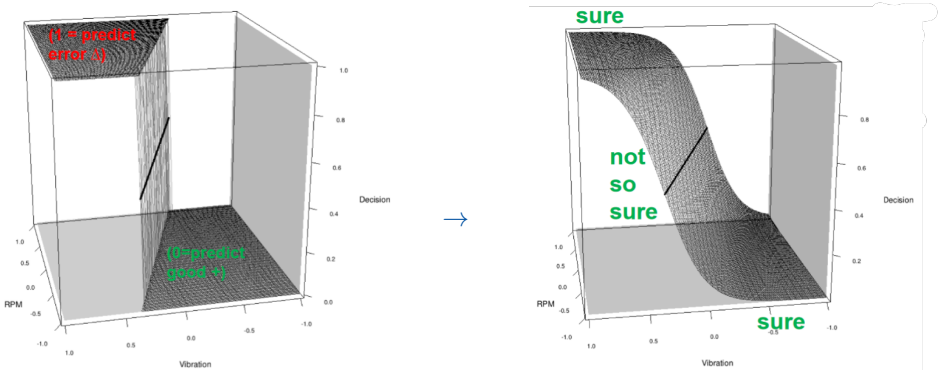
$$\frac{d}{dx} logistic(x) = logistic(x) \cdot \big(1 - logistic(x)\big)$$

Figure 4.4: Logistic function

- Here, $e = 2.7182818\cdots$ is Eulers number
- Any value is mapped on a value between 0 and 1
  - $logistic(0) = 0.5$, $logistic(-\infty) = 0$, $logistic(+\infty) = 1$
- 0 and 1 are quickly approached, therefore we can use it as a "smooth binary value"

Logistic regression

So now, we can use **logistic regression** instead of a hard 0/1-decision.



$$\mathbb{M}_{\mathbf{w}}(\mathbf{d}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{d} \geq 0 \\ 0 & \text{otherwise} \end{cases} \qquad \mathbb{M}_{\mathbf{w}}(\mathbf{d}) = \mathit{logistic}(\mathbf{w} \cdot \mathbf{d})$$

Figure 4.5: Difference 0/1 and logistic

The probabilistic interpretation looks as follows:

- $P[t =' \text{faulty}' \mid \mathbf{d}] = \mathbb{M}_{\mathbf{w}}(\mathbf{d})$
- $P[t =' \text{good}' \mid \mathbf{d}] = 1 - \mathbb{M}_{\mathbf{w}}(\mathbf{d})$
- And the system is more sure about the decision, the further it is away from the separating line.

4.6 Extensions (non-linear and multinomial)

Non-linear relationship

Non-linear functions that can't be separated by a line can still be handled by using linear machinery. The basic idea for that is to transform the data in advance, leading to the following formula:

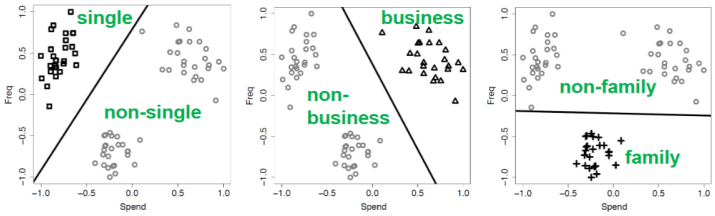
$$\mathbb{M}_{\mathbf{w}}(\mathbf{d}) = \sum_{k=0}^{b} \mathbf{w}[k] \ \phi_k(\mathbf{d})$$

- Data in non-linear relationships is transformed *before* applying the linear machinery.
- For that, apply "basis functions" ϕ_k like $\phi_0(x) = 1$, $\phi_1(x) = x$, $\phi_2(x) = x^2$, etc.

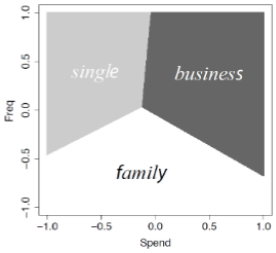
Multinomial regression

For the case of **multinomial regression**, we look at the problem where categorical data is not binary. As a solution for n possible classes, we can split the decision into n models (per class i one model, classifying as "is class i" or "is not class i"). Those can then be combined back again into one model.

(a) Original problem



(b) Binary classification



(c) Resulting model (multinomial classification)

Figure 4.6: Combining binary (one-versus-rest) models to multinomial regression

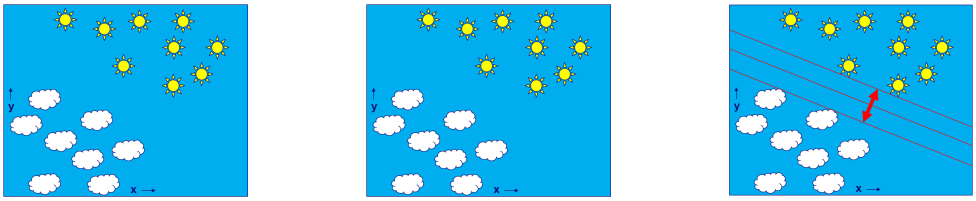
5 Support vector machine (SVM)

5.1 Basic functionality

Let's suppose we have the following dataset where

- Our target feature has two possible classes: sun and cloud.
- We have two descriptive features: x and y

The data set is separable, and our goal is now to find the best hyperplane (a line here) separating the two classes.



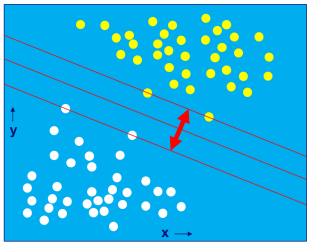
(a) Original data set (b) Proposal separating lines (c) Line with margin

Figure 5.1: Example for SVM

We need to determine which proposal lines are better than others. The idea to determine this is that the degrees of freedom become less when the separating hyperplane gets thicker. The best hyperplane is:

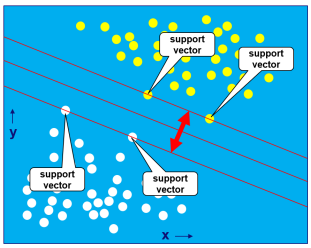
- The best-separating hyperplane is the one fitting between the data points with the maximal thickness (= safety margin).
- Therefore, SVMs also tend to be more robust than logistic regression.

Consider now a more abstract data set:



(a) Data set with hyperplane and safety margin

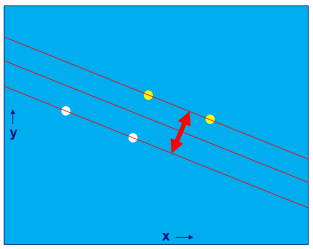
We have instances that are (atomic) points in an n-dimensional space.



(b) Support vectors

Support vector

Support vectors (SV) are the boundary points → only those matter.



(c) Reduced problem

Quadratic programming

Define a **QP** problem (quadratic programming) to make (compared to logistic regression):

- Things more efficient
- It less sensitive to outliers
- Reduce the risk of overfitting

But, there are applications where logistic regression works better.

Figure 5.2: SVM basic idea

The dimension of the separating hyperplane is always $n-1$ where n is the dimension of the space.

- One descriptive feature → hyperplane is a point (of dimension $0 = n-1$)
- Two descriptive features → hyperplane is a line (of dimension $1 = n-1$)
- Three descriptive features → hyperplane is a plane (of dimension $2 = n-1$)
- ...

For $n > 3$, the problem can't be visualized anymore, but SVMs also work for very large n.

- This is in contrast to logistic regression, which gets too complex at large n

Hyperplane

The technical definition of the **hyperplane** is as follows:

$$\vec{\mathbf{w}} \cdot \vec{\mathbf{x}} + b = \underbrace{\mathbf{w}_1\mathbf{x}_1 + \mathbf{w}_2\mathbf{x}_2 + \cdots + \mathbf{w}_n\mathbf{x}_n}_{\substack{\vec{\mathbf{x}}=(x_1,x_2,\ldots,x_n) \\ \vec{\mathbf{w}}=(w_1,w_2,\ldots,w_n)}} + b = 0$$

At least one of the weights is non-zero. The hyperplane then splits the n-dimensional into two disjoint parts:

- $\vec{\mathbf{w}} \cdot \vec{\mathbf{x}} + b \geq 0$
- $\vec{\mathbf{w}} \cdot \vec{\mathbf{x}} + b < 0$

There are some cases where the dataset is not separable.

- This can be due to **outliers**. A solution for this problem is to relax the problem by allowing violating instances $\rightarrow$ but they then add a penalty.
- Alternatively, there can also be a **structural** problem (not linearly separable in n-dimensional space). The idea to solve this problem is to try whether the problem is linearly separable in a higher-dimensional space, which then also includes feature design.

We need to make some further definitions to completely formalize the SVM approach. We'll first look at **vectors** like $\vec{\mathbf{x}} = (x_1, x_2, \ldots, x_n)$. Vector

- The vector **length** is defined as Vector length

$$\| \vec{\mathbf{x}} \| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

- The vector **direction** is the normalized vector of length one, so Vector direction

$$\vec{\mathbf{w}} = \frac{\vec{\mathbf{x}}}{\| \vec{\mathbf{x}} \|} = \left(\frac{x_1}{\| \vec{\mathbf{x}} \|}, \frac{x_2}{\| \vec{\mathbf{x}} \|}, \cdots, \frac{x_n}{\| \vec{\mathbf{x}} \|} \right)$$

- To extend the length of a vector without changing its direction, multiply the vector with a constant

$$q\vec{\mathbf{x}} = (q \cdot x_1, q \cdot x_2, \ldots, q \cdot x_n)$$

- The dot product on the other hand allows the multiplication of two vectors:

$$\vec{\mathbf{x}} \cdot \vec{\mathbf{y}} = \sum_{i=1}^{n} x_i y_i = \underbrace{\| \vec{\mathbf{x}} \| \cdot \| \vec{\mathbf{y}} \| \cdot \cos\theta}_{\text{with angle } \theta \text{ between } x,y} \quad \in \mathbb{R}$$

  - The dot product is heavily influenced by the angle between the two vectors and therefore also tells their linear dependency.
  - For orthogonal vectors $\cos\theta = 0$, for vectors with the same direction $\cos\theta = 1$ and for angles in the opposite direction $\cos\theta = -1$

Now, let's look deeper into the definition of a hyperplane. Our hyperplane is defined by

$$\vec{\mathbf{w}} \cdot \vec{\mathbf{x}} + b = 0 \iff \sum_{i=1}^{n} w_i x_i + b = 0$$

where

- $\vec{\mathbf{x}} = (x_1, x_2, \ldots, x_n)$ are free variables,
- $\vec{\mathbf{w}} = (w_1, w_2, \ldots, w_n)$ is the direction vector of the hyperplane (no-zero),
- And the hyperplane is fully defined by $\vec{\mathbf{w}}$ and b

Further interesting to see:

$$\text{For } q \in \mathbb{R} \setminus \{0\}: \quad \vec{\mathbf{w}} \cdot \vec{\mathbf{x}} + b = 0 \iff \sum_{i=1}^{n} w_i x_i + b = 0$$
$$\iff \quad q\vec{\mathbf{w}} \cdot \vec{\mathbf{x}} + qb = 0 \iff \sum_{i=1}^{n} q w_i x_i + qb = 0$$

The hyperplane splits a set of data points in the following way. Assume we have N data points $X_N = \{\vec{\mathbf{x}}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,n}) \,|\, i \in N\}$. Then, for $\vec{\mathbf{x}}_i$ calculate $f(\vec{\mathbf{x}}_i) := \vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_i + b = \sum_{j=1}^{n} w_j x_{i,j} + b$, and

$$\text{Classifiy as } \begin{matrix}\text{positive} \\ \text{negative}\end{matrix} \text{ if } f(\vec{\mathbf{x}}_i) \begin{matrix}\geq 0 \\ < 0\end{matrix}$$

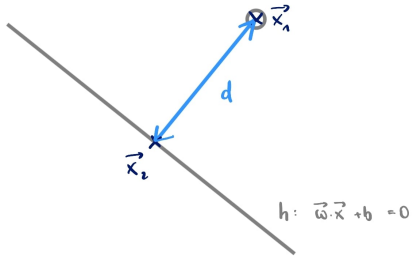
Note the ongoing confusion of whether above or below the hyperplane is defined as positive, e.g. by multiplying $\vec{\mathbf{w}}$ and b with $q = -1$

Perfectly separating hyperplane

By splitting the data as described above, we introduce two possible classes for each $\vec{\mathbf{x}}_i$, namely $y_i \in \{-1, 1\}$. If our hyperplane $(\vec{\mathbf{w}}, b)$ **separates** a dataset X_N **perfectly**, then:

$$\text{For all instances } (\vec{\mathbf{x}}_i, y_i): \quad \begin{matrix} y_i = +1 & \implies & f(\vec{\mathbf{x}}_i) \geq 0 \\ y_i = -1 & \implies & f(\vec{\mathbf{x}}_i) < 0 \end{matrix}$$

But actually, there might be many "perfect" hyperplanes. The one we would like to pick is the one that maximizes the distance of the hyperplane to the nearest point. For that, we of course first need to define the **distance between a point and a hyperplane**. Assume, we have the following setting:



Assume any point $\vec{\mathbf{x}}_1$ and hyperplane $(\vec{\mathbf{w}}, b)$

- $\vec{\mathbf{x}}_2$ is closest point on the hyperplane
- d as distance between $\vec{\mathbf{x}}_1$ and the hyperplane, defined as the distance between the two points

Figure 5.3: Distance between a point and the hyperplane

As can be seen in figure 5.3, we have the closest point defined as the point that can be found by following an orthogonal path and seeing which point is on the hyperplane:

$$\underbrace{\vec{\mathbf{x}}_1 = \vec{\mathbf{x}}_2 + \lambda\vec{\mathbf{w}}}_{\text{orthogonal path}} \text{ and } \underbrace{\vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_2 + b = 0}_{\vec{\mathbf{x}}_2 \text{ is on hyperplane}}$$
$$\implies \quad \vec{\mathbf{w}} \cdot (\vec{\mathbf{x}}_1 - \lambda\vec{\mathbf{w}}) + b = 0$$
$$\implies \quad d = \|\lambda\vec{\mathbf{w}}\|$$

Next, we try to get rid of the λ in the computation of $d = \|\lambda\vec{\mathbf{w}}\|$. For that, just rewrite

the formula for the calculation of $\vec{x}_2$:

$$\begin{aligned}
& \vec{w} \cdot (\vec{x}_1 - \lambda \vec{w}) + b = 0 \\
\Longrightarrow \quad & \vec{w} \cdot \vec{x}_1 + b = \lambda \vec{w} \cdot \vec{w} = \lambda \|\vec{w}\|^2 \\
\Longrightarrow \quad & \lambda = \frac{\vec{w} \cdot \vec{x}_1 + b}{\|\vec{w}\|^2} \\
\Longrightarrow \quad & d = \left\| \left(\frac{\vec{w} \cdot \vec{x}_1 + b}{\|\vec{w}\|^2} \right) \cdot \vec{w} \right\| \\
& = \left| \frac{\vec{w} \cdot \vec{x}_1 + b}{\|\vec{w}\|^2} \right| \|\vec{w}\| = \left| \frac{\vec{w} \cdot \vec{x}_1 + b}{\|\vec{w}\|} \right|
\end{aligned}$$

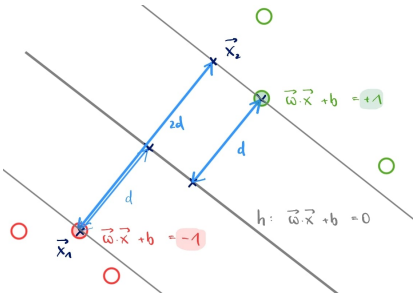
So finally, we can formulate how to calculate the **optimal hyperplane**:

Optimal hyperplane

$$\begin{aligned}
&\text{Given } \left\{ (\vec{x}_i, y_i) \in \mathbb{R}^n \times \{-1, 1\} \,\middle|\, 1 \le i \le m \right\} \\
&\quad \text{Find } (\vec{w}, b) \text{ such that for any } 1 \le i \le m: \quad \begin{array}{l} \text{If } y_i = +1 \quad \text{, then } \vec{w} \cdot \vec{x}_i + b \ge 0 \\ \text{If } y_i = -1 \quad \text{, then } \vec{w} \cdot \vec{x}_i + b < 0 \end{array} \\
&\quad\quad \text{And maximize } \min_{1 \le i < m} d_i \text{ with: } \quad d_i = \left| \frac{\vec{w} \cdot \vec{x}_i + b}{\|\vec{w}\|} \right|
\end{aligned}$$

Alternatively, we could normalize the distance to one, such that the resulting situation is as depicted in 5.4.

- This rescaling can be performed since the hyperplane formulation doesn't change when multiplying with a constant q.
- The support vectors yield the result $\vec{w} \cdot \vec{x}_i + b = \pm 1$.



The support vectors are on the lines
(positive and negative support vectors)

Figure 5.4: Normalized distance

Also, the classification is adjusted to

$$\left.\begin{array}{l} \text{If } y_i = +1 \quad \text{, then } \vec{w} \cdot \vec{x}_i + b \ge +1 \\ \text{If } y_i = -1 \quad \text{, then } \vec{w} \cdot \vec{x}_i + b \le -1 \end{array}\right\} \rightarrow \quad \begin{array}{l} \text{can be combined to:} \\ \forall i : y_i(\vec{w} \cdot \vec{x}_i + b) \ge 1 \end{array}$$

Optimal hyperplane normalized

Our goal is still to maximize the smallest $d = \|\lambda \vec{w}\|$. In our alternative formulation, we want to formalize this goal only with the help of the shifted hyperplanes $\vec{x} \cdot \vec{x} + b = \pm 1$:

- Choose an arbitrary support vector $\vec{x}_1$ (let's say on $\vec{x} \cdot \vec{x} + b = -1$)
- Now take the corresponding $\vec{x}_2 = \vec{x}_1 + 2\lambda \vec{w}$ on the other support-vector-line (so $\vec{w} \cdot \vec{x} + b = +1$)

- This leads to the following equations to be solved:

$$\begin{aligned} \vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_1 + b &= -1 \\ \vec{\mathbf{w}} \cdot (\vec{\mathbf{x}}_1 + 2\lambda\vec{\mathbf{w}}) + b &= +1 \end{aligned}$$

- This can be further readjusted to:

$$\begin{aligned} \implies \quad & \underbrace{\vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_1 + b}_{=-1} + \vec{\mathbf{w}} \cdot 2\lambda\vec{\mathbf{w}} = +1 \\ \implies \quad & \vec{\mathbf{w}} \cdot 2\lambda\vec{\mathbf{w}} = +2 \\ \implies \quad & \lambda = \frac{1}{\vec{\mathbf{w}} \cdot \vec{\mathbf{w}}} = \frac{1}{\|\vec{\mathbf{w}}\|^2} \end{aligned}$$

- So, we can replace:

$$\begin{aligned} &\text{Maximize } d = \|\lambda\vec{\mathbf{w}}\| = \frac{1}{\|\vec{\mathbf{w}}\|^2}\|\vec{\mathbf{w}}\| = \frac{1}{\|\vec{\mathbf{w}}\|} \\ &\text{Minimize } \|\vec{\mathbf{w}}\| \text{ under the constraints stated before} \end{aligned}$$

So, the reformulated problem is:

Reformulated: SVM problem

$$\begin{gathered} \text{Given a set of } m \text{ instances } \left\{(\vec{\mathbf{x}}_i, y_i) \in \mathbb{R}^n \times \{-1, 1\} \,\middle|\, 1 \leq i \leq m\right\} \\ \min_{\vec{\mathbf{w}}, b} \|\vec{\mathbf{w}}\| \text{ s.t. } \forall i : y_i(\vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_i + b) \geq 1 \end{gathered}$$

For technical reasons, a common variant of this problem is the convex quadratic optimization problem

Convex quadratic optimization problem

$$\begin{gathered} \text{Given a set of } m \text{ instances } \left\{(\vec{\mathbf{x}}_i, y_i) \in \mathbb{R}^n \times \{-1, 1\} \,\middle|\, 1 \leq i \leq m\right\} \\ \min_{\vec{\mathbf{w}}, b} \frac{1}{2}\|\vec{\mathbf{w}}\|^2 \text{ s.t. } \forall i : y_i(\vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_i + b) \geq 1 \end{gathered}$$

- This problem is easier to solve than the original linear formulation (maximizing $\min_{1 \leq i < m} d_i$)
- The concrete solution or techniques for finding solutions are not covered here, but assume they exist

Another reformulation of the problem is the Wolfe dual Lagrangian problem which is important for the "kernel trick":

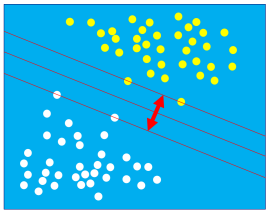
Wolfe dual Lagrangian problem

$$\begin{gathered} \text{For } \vec{\mathbf{w}} = \sum_{i=1}^{m} \alpha_i y_i \vec{\mathbf{x}}_i \text{ and } \alpha_i \geq 0, i = 1, \ldots, m \\ \begin{aligned} \max_\alpha \quad & \textstyle\sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \vec{\mathbf{x}}_i \vec{\mathbf{x}}_j \\ \text{subject to} \quad & \textstyle\sum_{i=1}^m \alpha_i y_i = 0 \end{aligned} \end{gathered}$$

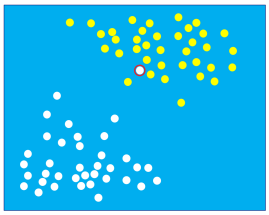
5.2 Soft-margin SVMs

So far, we looked at the general problem formulation of SVMs when the dataset was separable. Next, we're gonna look at how to handle datasets where no reasonable hyperplane can be found to separate the instances, even after removing outliers.

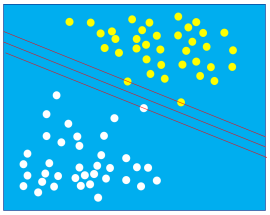
To see, what the problem is, consider the examples from figure 5.5.



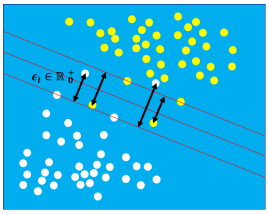
(a) Nicely separable data



(b) One outlier destroying separability



(c) Outlier reducing margin



(d) Reasonable hyperplane tolerating outliers
$\rightarrow$ add goal of minimizing slack variables

Figure 5.5: Separability of datasets

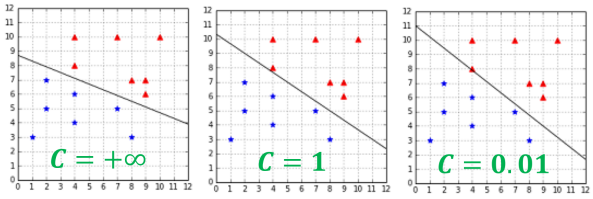
So, the reformulated problem is the following:

Given a set of m instances $\{(\vec{\mathbf{x}}_i, y_i) \in \mathbb{R}^n \times \{-1, 1\} \mid 1 \leq i \leq m\}$ Soft-margin SVM problem

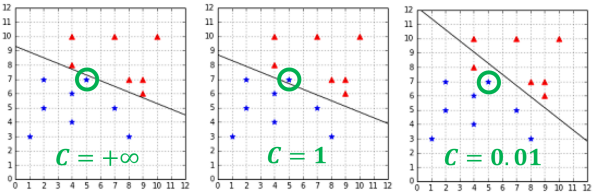
$$\min_{\vec{\mathbf{w}}, b, \epsilon} \frac{1}{2}\|\vec{\mathbf{w}}\|^2 + C\sum_{i=1}^{m} \epsilon_i \text{ s.t. } \forall i : y_i(\vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_i + b) \geq 1 - \epsilon_i \text{ where } \epsilon_i \geq 0$$

The role of C determines how much we want to restrict tolerance for outliers.

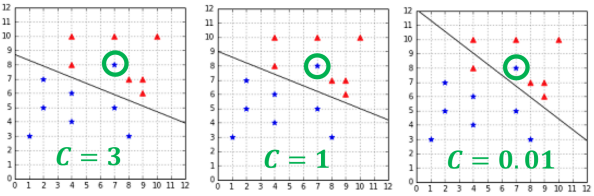
- For a high C, we have a strict formulation of our problem, so no outlier tolerance.
- For a low C, we have a more flexible formulation of the problem, so outliers are tolerated.
- Consider 5.6 to see the effect of C on different datasets.



(a) Separable dataset



(b) One outlier, but still separable dataset



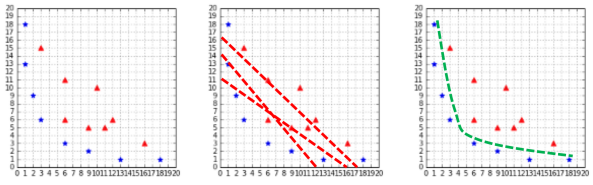
(c) Not separable dataset
Especially for $C = +\infty$: no solution of SVM exists for this problem

Figure 5.6: Role of C for soft-margin SVM

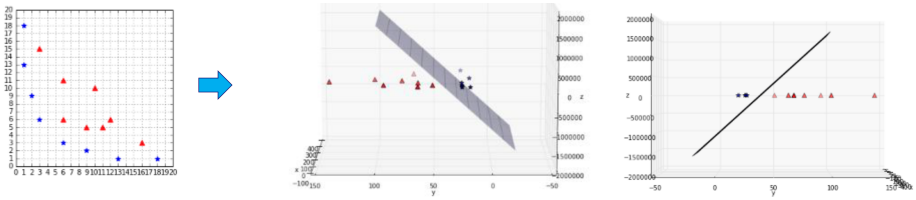
5.3 Non-linear decision boundaries

Next, we're considering non-linear separation, where our dataset can't be separated by a linear hyperplane, but by some other non-linear decision boundary. Consider the example in 5.7. Our idea to solve this separation is to **lift the number of dimensions**.

Lift number of dimensions



(a) Non-linear decision boundary



(b) Lift from $n = 2$ to $n = 3$

Figure 5.7: Non-linear separation

The mapping to lift the dimensions is similar to the one introduced for regression. So, we have the updated problem formulation:

Soft-margin, dimension-lifted SVM problem

$$\text{Given a set of } m \text{ instances } \left\{(\vec{\mathbf{x}}_i, y_i) \in \mathbb{R}^n \times \{-1, 1\} \,\middle|\, 1 \leq i \leq m\right\}$$
$$\text{Find a mapping } \phi \in \mathbb{R}^n \to \mathbb{R}^q \text{ with } q > n$$
$$\text{E.g.: } \phi_0(x) = 1, \phi_1(x) = x, \phi_2(x) = x^2, \ldots$$
$$\min_{\vec{\mathbf{w}}, b, \epsilon} \frac{1}{2}\|\vec{\mathbf{w}}\|^2 + C\sum_{i=1}^{m} \epsilon_i \text{ s.t. } \forall i : y_i(\vec{\mathbf{w}} \cdot \phi(\vec{\mathbf{x}}_i) + b) \geq 1 - \epsilon_i \text{ where } \epsilon_i \geq 0$$

For our example 5.7, the mapping function was formulated as:

$$\phi \in \mathbb{R}^2 \to \mathbb{R}^3$$
$$\phi(x_1, x_2) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)$$

5.4 Using kernels

Let's look again at the **Wolfe dual Lagrangian problem**, but this time applied to the dimension-lifting trick from before.

Wolfe dual Lagrangian problem with kernel

$$\text{For } \vec{\mathbf{w}} = \sum_{i=1}^{m} \alpha_i y_i \vec{\mathbf{x}}_i \text{ and } \alpha_i \geq 0, i = 1, \ldots, m$$
$$\max_\alpha \quad \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \underbrace{K(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j)}_{=\phi(\vec{\mathbf{x}}_i)\cdot\phi(\vec{\mathbf{x}}_j)}$$
$$\text{subject to} \quad \sum_{i=1}^m \alpha_i y_i = 0$$

- Once again, we have $\phi \in \mathbb{R}^n \to \mathbb{R}^q$ with $q > n$
- K is our **kernel function** — Kernel function
  - It efficiently computes $\phi(\vec{\mathbf{x}}_i) \cdot \phi(\vec{\mathbf{x}}_j)$
  - S.t., the actual mapping mappings $\phi(\vec{\mathbf{x}}_i)$ and $\phi(\vec{\mathbf{x}}_j)$ as well as their dot product don't need to be computed

Consider the kernel function for our previous example 5.7.

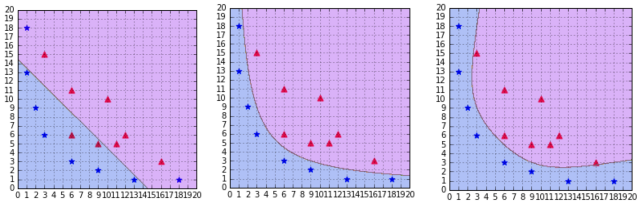
$$\phi(x_1, x_2) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)$$
$$\implies \phi(x_{i/j,1}, x_{i/j2}) = \left(x_{i/j,1}^2, \sqrt{2}x_{i/j,1}x_{i/j,2}, x_{i/j,2}^2\right)$$
$$\implies K(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j) = \phi(\vec{\mathbf{x}}_i) \cdot \phi(\vec{\mathbf{x}}_j) = \underbrace{x_{i,1}^2 x_{j,1}^2}_{(x_{i,1}x_{j,1})^2} + \underbrace{2x_{i,1}x_{i,2}x_{j,1}x_{j,2}}_{2(x_{i,1}x_{j,1})(x_{i,2}x_{j,2})} + \underbrace{x_{i,2}^2 x_{j,2}^2}_{(x_{i,2}x_{j,2})^2}$$
$$= (x_{i,1}x_{j,1} + x_{i,2}x_{j,2})^2$$

- So, for calculating $\phi(\vec{\mathbf{x}}_i)\cdot\phi(\vec{\mathbf{x}}_j)$, we need 10 multiplication-operations and 2 addition-operations,
- Whereas for calculating $K(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j)$, we need 3 multiplication-operations and 1 addition-operation.

In the following, we'll see some kernel functions:

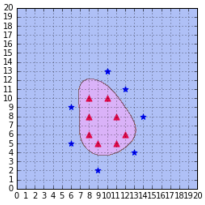
- We can use **polynomial kernels** with parameters c (constant) and d (degree, remember the danger of overfitting with increasing degree).

$$K(\vec{\mathbf{x}}_i, \vec{\mathbf{x}}_j) = \underbrace{\phi(\vec{\mathbf{x}}_i) \cdot \phi(\vec{\mathbf{x}}_j)}_{\phi \text{ may have a variable (large) number of dimensions}} = (\vec{\mathbf{x}}_i \cdot \vec{\mathbf{x}}_j + c)^d$$



$d = 1$ $d = 2$ $d = 6$

- Alternatively, there are e.g. **circular kernels**, since some data can't be separated by any polynomial.



6 Naive Bayesian Classification

Next, we're gonna look at a classification based on stochastical formulas developed by Thomas Bayes (1701-1761, English statistician and philosopher). More concretely, the classification is based on Bayes' theorem, often used in probability-based learning.

First, we'll lay down some basic statistics:

$$\begin{aligned}
A, \dots, Z &: \text{Some event (can either be true or false)} \\
P(X) &: \text{Probability that } X \text{ holds} \\
P(X, Y) &: \text{Probability that both } X \text{ and } Y \text{ hold} \\
P(X|Y) &: \text{Probability that } X \text{ holds given } Y \text{ (conditional probability)} \\
P(\neg X) &= 1 - P(X) \\
P(\neg X|Y) &= 1 - P(X|Y) \\
P(Y) &= P(Y|X)P(X) + P(Y|\neg X)P(\neg X)
\end{aligned}$$

Further, we have the product and chain rule:

$$\begin{aligned}
P(X, Y) &= P(X|Y) \cdot P(Y) \\
P(A, B, \dots, Y, Z) &= P(A, B, \dots, Y|Z) \cdot P(Z) = P(A|B, \dots, Y, Z) \cdots P(Y|Z) \cdot P(Z)
\end{aligned}$$

If X and Y are independent, so $P(X|Y) = P(X)$, it further holds[4]:

$$P(X, Y) = P(X) \cdot P(Y)$$

And finally, we have **Bayes' theorem**, and so the generalized one, which can be simply derived from the product rule: Bayes' Theorem

$$P(Y|X) = \frac{P(X|Y) \cdot P(Y)}{P(X)} P(Y = y|x_1, \dots, x_m) \quad = \frac{P(x_1, \dots, x_m|Y = y) \cdot P(Y = y)}{P(x_1, \dots, x_m)}$$

We can now apply the theorem to classification, by setting X as the event that the target feature has a specific value and Y that the descriptive features have specific values. The probabilities can then be estimated trivially. Formally, we have the Bayesian **maximize a posteriori (MAP)** prediction model: MAP

$$\begin{aligned}
\mathbb{M}_{MAP}(\vec{\mathbf{x}}) &= \arg\max_{\vec{\mathbf{y}}} P[y|x_1, \dots, x_m] \\
&= \arg\max_{\vec{\mathbf{y}}} \frac{P(x_1, \dots, x_m|y) \cdot P(y)}{P(x_1, \dots, x_m)}
\end{aligned}$$

Or without normalization:

$$\mathbb{M}_{MAP}(\vec{\mathbf{x}}) = \arg\max_{\vec{\mathbf{y}}} P(x_1, \dots, x_m|y) \cdot P(y)$$

To avoid overfitting, and solve some computational problems due to the curse of dimensionality, we can assume independence, which then results in the **Naive Bayes'**

[4] Extendable for any number of events

Classifier:

$$\mathbb{M}(\vec{\mathbf{x}}) = \arg\max_{\vec{\mathbf{y}}} \left(\prod_{i=1}^{m} P[x_i|y] \right) \cdot P(y)$$

- Many combinations of features don't appear in the training data, but we can't conclude that these cannot happen.
- We can use this classifier, to approximate them.

7 Neural networks

So far, we have looked at different **supervised learning** techniques:

- Decision trees (first categorical, then numerical)
- Regression (first numerical, then categorical)
- Support vector machines (SMV)
- Naive Bayesian classifier

All those techniques have the following things in common:

- They try to learn a **function predicting** a **target** feature in terms of its descriptive features.

$$f(\underbrace{\vec{\mathbf{x}}}_{\text{descriptive features}}) := \underbrace{\vec{\mathbf{y}}}_{\text{target feature}}$$

- Some way of measuring the **error** is provided, e.g. the sum of squared errors, or the number of misclassifications.

$$m_{error} : \{(\underbrace{f(\vec{\mathbf{x}}_i)}_{\text{predicted label}}, \underbrace{\vec{y}_i}_{\text{correct label}})\} \mapsto d \in \mathbb{R}$$

- Learning is now based on **training data**. The evaluation then requires **test data** (unseen) to address the problem of overfitting.

$$\mathcal{D}_{train}, \mathcal{D}_{test} \subseteq \{(\underbrace{\vec{\mathbf{x}}_i}_{\text{input}}, \underbrace{\vec{y}_i}_{\text{correct label}})\}, \quad \mathcal{D}_{train} \cap \mathcal{D}_{test} = \emptyset$$

For **classification** and specifically linear regression and SVMs, we derived the following formula:

$$y_{\vec{\mathbf{w}}}(\vec{\mathbf{x}}) = f\left(\sum_{j=1}^{M} \vec{\mathbf{w}}_j \phi_j(\vec{\mathbf{x}}) \right)$$

- $\vec{\mathbf{x}}$ is the input vector containing the descriptive features
- $\vec{\mathbf{w}}$ is our weights vector
- The input vector can be lifted to a higher dimension by a mapping ϕ:
  - Simple linear classification: $\vec{\mathbf{x}}$ can be used directly
  - Nonlinear classification: apply feature space mapping or the "kernel trick"
    $\rightarrow$ requires lots of domain knowledge
    $\rightarrow$ increases the number of features M

- Finally, we see the actual categorizer f, which is either:
  - A simple threshold-based function with returning 0 or 1, or
  - A more sophisticated function like the sigmoid function (is used in logistic regression)

We can abstract that further to the following image:

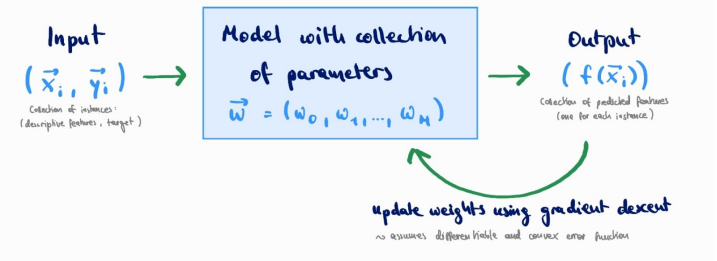


Figure 7.1: Abstraction on supervised learning

- **Gradient descent** is a basic principle to iteratively reduce the error by walking down the hill in the steepest direction.
- Important is the choice of the step size. If it's too small, the convergence is slow. If it is too large, a risk of overshooting or even divergence arises.

Now, the topic of this chapter comes into play: why do we need **neural networks**?

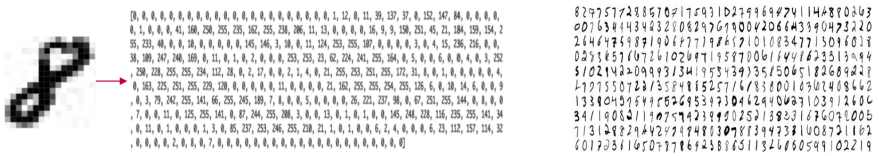
- The input for our classification problem can be anything, e.g. text, sound, images, videos, etc.
- This would mean, our classifier can no longer be described by a simple (linear) function.
- Our traditional linear approaches don't work anymore. SVM already tried to address this problem by introducing the mapping $\phi(\vec{\mathbf{x}})$ to lift the number of dimensions, but this mapping needed to be manually constructed.
- In the NN approach: $\phi(\vec{\mathbf{x}})$ may be based on other layers and can be learned.

Our before-seen function is therefore lifted to the following:

$$y_{\vec{\mathbf{w}}}(\vec{\mathbf{x}})[k] = f\left(\sum_{j=1}^{M} w_{jk}^{(2)} \underbrace{h\left(\sum_{i=1}^{D} w_{ij}^{(1)} x_i + w_{0j}^{(1)}\right)}_{\text{"}\phi_j(\vec{\mathbf{x}})\text{" is now a network}} + w_{0k}^{(2)}\right)$$

- We have input $\vec{\mathbf{x}}$ with D dimensions or features.
- The network uses the activation functions f and h.

Consider the classification of images with automatic dog detection. For that, many labeled samples (dog and non-dogs) are provided. The samples are pictures, so for the computer just a collection of pixels. This results in a huge number of features. The same can be seen for the example in 7.2. Generally, as the input for our NNs, we're only gonna consider **unstructured data** such as images, text, sound, or video.



Set of features

Training dataset

$$\text{Image} = \text{grid of numbers representing the darkness of each pixel (grayscale)}$$

$$\text{Numbers for MNIST are stored as } 18 \times 18 \text{ pixel images} \implies \text{Array of } 324 \text{ numbers}$$

Figure 7.2: MNIST dataset example (number recognition classifier)

After we saw the intuition of why we need NNs, what they roughly are, and some example applications, we'll now take a look at what an Artificial NN really is.

- Generally speaking, it is a computing system inspired by, but not identical to, biological NNs.
  - Since there are many differences between real NNs and artificial ones, the terms are sometimes criticized.
- It learns to perform tasks by considering examples without being explicitly programmed.
- Artificial NNs are collections of connected artificial neurons.
  - The connections correspond to weights that can be updated to make the error on the training data smaller.

7.1 Historical background

Throughout the history of developing NNs (see 7.3), we can see two paradigms for AI:

1. Logic-inspired based on **reasoning** "Good old-fashioned AI"
   - Symbolic rules over symbolic expressions
   - Programmed using unambiguous language

   $\rightsquigarrow$ $1943 - 2005$: beat NNs with various other methods
2. Biologically-inspired based on **learning**
   - Large vectors representing neural activities
   - Vectors learned from data

   $\rightsquigarrow$ $2005 - 2010$: NNs show more promising results (improvements in backpropagation, more training material, computational power)

   $\rightsquigarrow$ > 2010 : major investments (Google, IBM, Apple etc. for Alexa, Siri, autonomous driving, ...)

   $\rightsquigarrow$ > 2018: high on political agenda

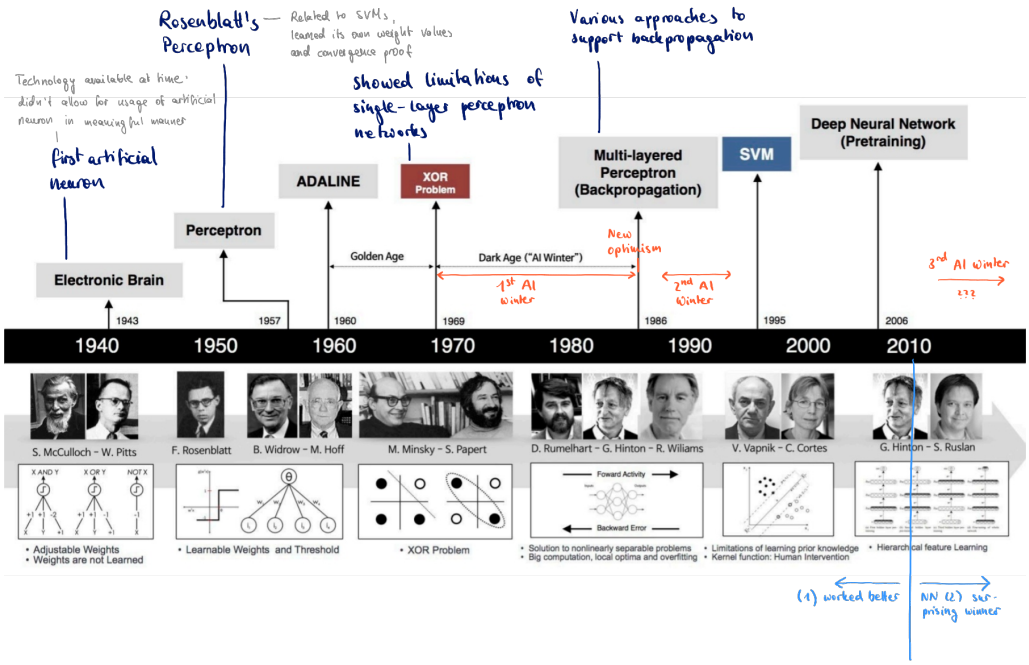


Figure 7.3: History of Neural Networks

In 2018, the Turing Award was given for the development of a convolutional NN, concretely the conceptual and engineering breakthroughs making deep neural networks a critical component of computing. Still, for a NN to work successfully, a lot of engineering and trial-and-error is necessary.

Pro and Con of NNs

- The main advantages of ANNs:
  - NNs are best at identifying patterns or trends, so they are well suited for:
    - Sales forecasting, logistics, customer behavior, security, medicine
    - Specific tasks: face recognition, traffic sign classification, sentiment analysis, diagnosis of hepatitis, speech recognition, hand-written text recognition, computer vision, pattern recognition, etc.
  - Can model complex (non-linear) functions
  - Generic and flexible, driven by data
  - Good performance on unseen noisy data
  - Can handle images, sound, text, video, etc.
  - After the model is learned: can be applied fast
- Main disadvantages:
  - Only works with lots of training examples
  - Time- and resource-consuming
  - Non-transparent (black-box, interpretation of hidden layers is difficult → keyword explainable AI)
    - What the Hidden layers are doing is feature extraction. However due to the nonlinearity, it can be hard to interpret what feature they're extracting,
  - Risk of overfitting

    - Large number of parameters allows more complex functions (with enough parameters, you can fit any data set exactly)
    - But they require lots of training data and are drawn to overfitting the model
    - So they "learn" the training data by heart, without abstracting
  - Performs worse on well-defined problems
  - Can be "hacked" (add noise, switch one pixel → wrong classification, etc.)

7.2 Human and Artificial Neurons

Now that we know some of the historical developments, we'll take a look at how the ideas to develop artificial NNs came up.

The "original" biological inspiration is the **Human brain** with around 100 billion neurons. The human brain learns in the way displayed in 7.4.

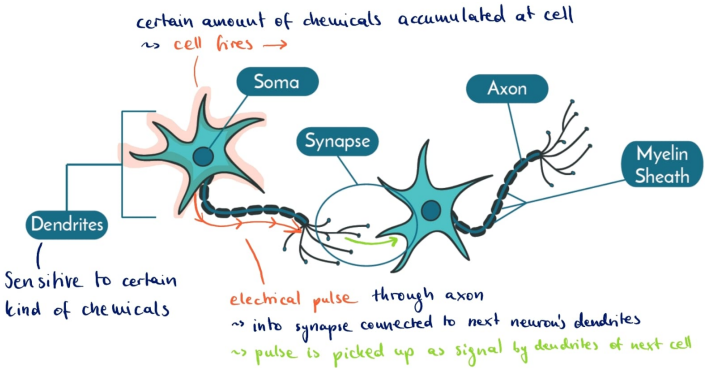


Figure 7.4: Biological neuron (learning)

So on a more abstract level: when a neuron receives excitatory input, learning occurs.

- The input needs to be sufficiently large compared to its inhibitory input, only then is a spike of electrical activity sent down the axon.
- Learning is then the change of the effectiveness of the synapses, and/or the influence of one neuron on other changes.

Single-layer/Rosenblatt's perceptron

This abstraction can be reduced to this simple (artificial) neuron shown in 7.5. This is also known as a **single-layer perceptron** which is the simplest feedforward neural network and only works for binary classification.

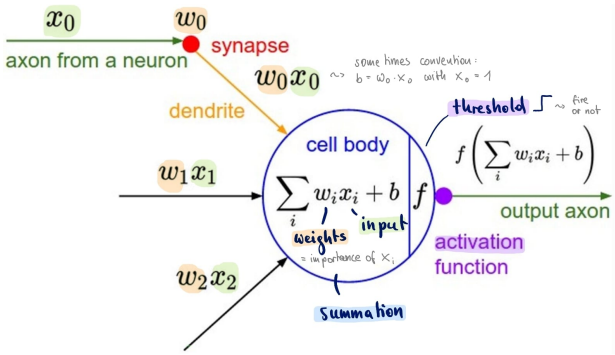


Figure 7.5: Simple artificial neuron (Rosenblatt's perceptron)

The **learning process** of this simple neuron is now the following:

- Randomly assign weights $w_i \in [0, 1]$
- Present inputs from training data $\vec{\mathbf{x}}$
- Get the output $\vec{\mathbf{y}} = f\left(\sum_i w_i x_i\right)$
- Compare the calculated output to the training label
  - $\rightsquigarrow$ nudge weights to get results towards the desired target output
  - Repeat until the error is small or the given number of epochs is completed

An important detail of the neuron, which needs to be set in the beginning and can't be changed throughout the learning process, is the **activation function**. The choice is free, but here are two typical choices: Activation function

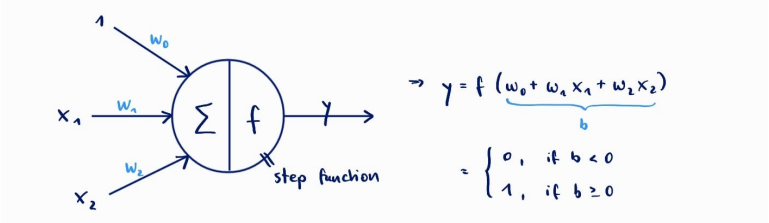
- Biological neurons use a threshold to decide when to fire. This simple **step function** can be imitated by Step function

$$f(x) = \begin{matrix} 0, & x < 0 \\ 1, & x \geq 0 \end{matrix}$$

- In NNs, the **sigmoid function** (just as in linear regression) is commonly used as the activation function f. Sigmoid function

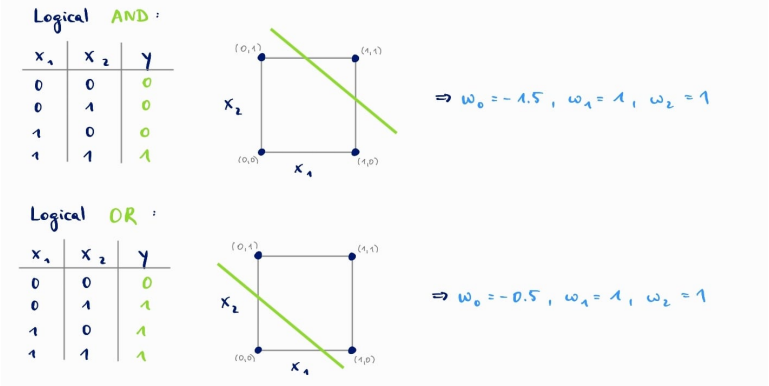
$$f(x) = sigmoid(x) = \frac{1}{1 + e^{-x}}$$

Finally, for the simple one-layer perceptrons, we're gonna look at some examples to see what artificial neurons can learn.



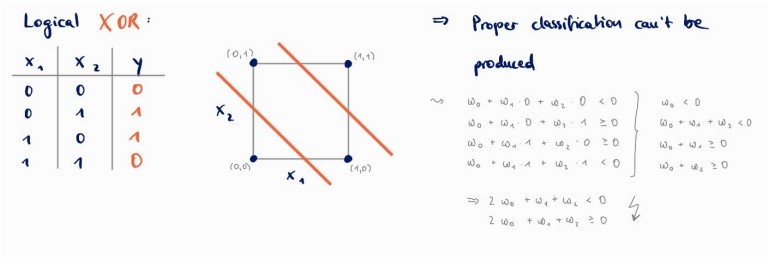
Simple NN architecture

AND- and OR-problem



Learnable problems

XOR-problem



Not-learnable problem

Figure 7.6: One-layer perceptron: Binary classification examples

The "XOR" example shows the limitations of single-layer perceptrons:

- Only a limited set of functions can be represented (e.g. even simple XOR not expressable)
- Decision boundaries must be hyperplanes (no non-linearity representable)
- Can only perfectly separate linearly separable data

7.3 Feedforwards Networks

Since the limitations of single-layer perceptrons are too strong, there is a need for more complex networks that also allow us to realize more complex problems. Different types of networks are depicted in 7.7.

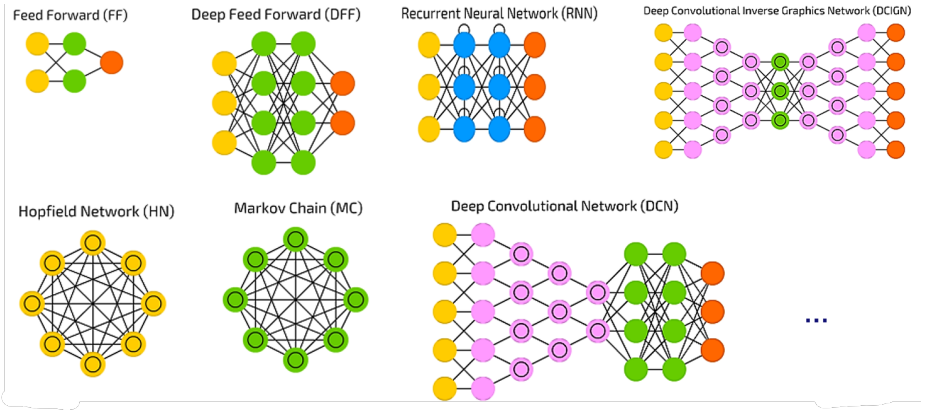


Figure 7.7: Different types and topologies of NNs

Convolutional Neural Network

One very often used architecture is the **convolutional NN** (CNN) as shown in 7.8. The intuitive idea is to have a hierarchy of visual elements that start by identifying edges using filters etc. and then move to more complex shapes. The extracted features then can be used for classification. CNNs are therefore best applied when the order of features matters, e.g. to successfully capture the spatial and temporal dependencies.

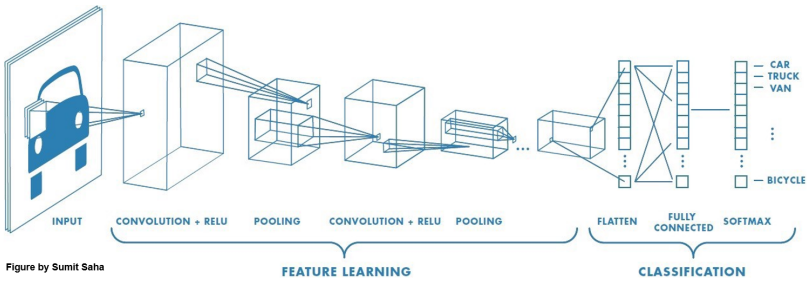


Figure 7.8: CNN architecture

Recurrent Neural Network

Another exemplary more complex architecture is the **recurrent NN** (RNN), also known as Long Short-Term Memory (LSTM). This network processes sequences of data such as speech or video. It can find the next-following most-fitting element for a sequence. The details of this network architecture are not discussed here.

- E.g.: "I lived in the Netherlands and speak perfectly Dutch"
- E.g.: "2, 4, 16, 32, 64, 128"

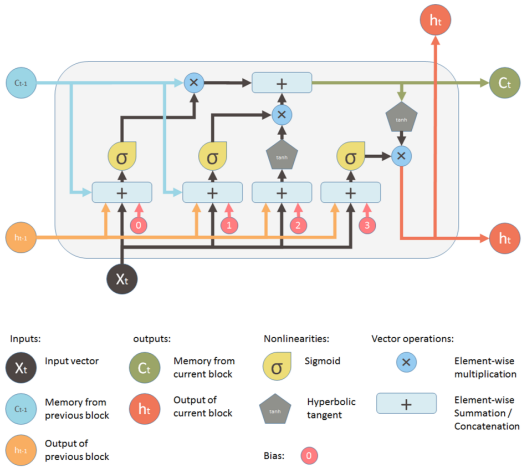


Figure 7.9: RNN architecture

Feedforward Neural Networks

For this course, we're only gonna look at **feedforward neural networks** (FNN), also called multi-layer perceptrons, that don't contain any loops[5] or special layers[6]. This means we have:

- Multiple simple perceptrons are arranged in a way that **layers** can be identified, and connections only exist to the next layer.
- The weights of the connection (in between two layers) can be changed or trained.
- The activation function (just as in a single-perceptron case) calculates whether the neuron fires or not.

To see, that FNNs really can express more functions than single-layer networks, we'll again consider the "XOR" example. As can be seen in 7.10. The result can be generalized, such that any boolean function with two input values can be represented by a two-layer perceptron.

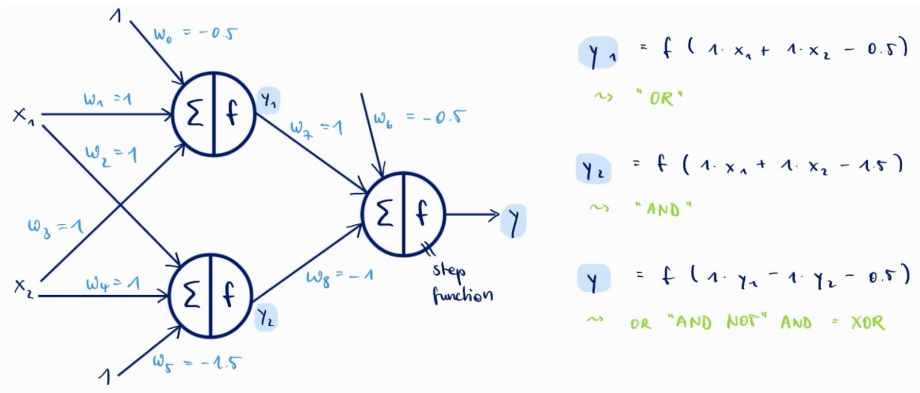


Figure 7.10: XOR with FNN

To lead us through the whole chapter, we'll introduce a general 2-layer FNN implementing

[5] Like in RNNs
[6] Like in CNNs

the function

$$y_k(\vec{\mathbf{x}}, \vec{\mathbf{w}}) = f\left(\sum_{j=0}^{M} w_{jk}^{(2)} \cdot \underbrace{h\left(\sum_{i=0}^{D} w_{ij}^{(1)} x_i\right)}_{=:z_j}\right), \forall k \in [N]$$

- Weights are notated as $w_{\text{from to}}^{(\text{layer})}$.
- f and h represent activation functions (can be different, but every layer is assumed to have the same activation function on all its neurons).
- We have input dimension D, M nodes in the hidden layer, and N output neurons.
- The hidden units with their activation functions can express non-linear functions.

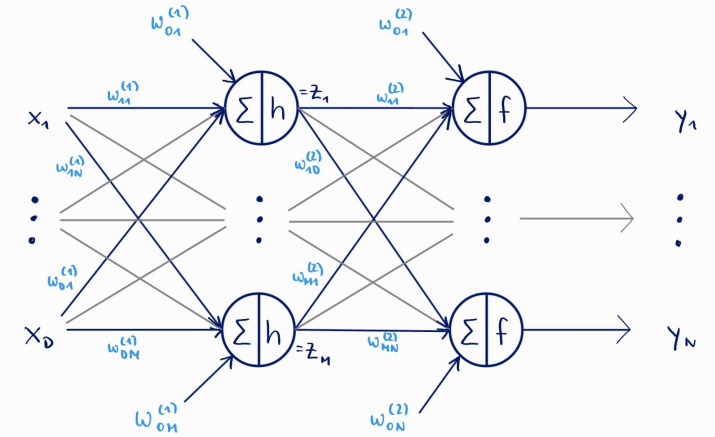


Figure 7.11: 2-layer FNN architecture

7.4 Network parameters

Generally, a **topology** of a network consists of the following network parameters that need to be decided before starting with training: Topology

- Number of **input** units (typically given)
- Number of **hidden** layers and number of neurons in each hidden layer (one or more layers)
- Number of **output** units (typically given)

Those parameters, but also generally how to train a network, raise the following questions:

- How are the inputs selected?
- How many hidden layers/neurons?
- How many neurons are in the output layer?
- How are the weights initialized? When and how are they updated?
- How many examples are (needed to be) in the training set?

We'll investigate those training parameters now in a bit more detail, starting with the **inputs**.

- Typically, we normalize our inputs such that the values fall in the range of $[0.0, 1.0]$

- Nominal or discrete-values attributes may be encoded s.t. there is one input unit per domain value
  - For that we can use "One Hot Encoding", which we already saw in a previous chapter.
  - E.g.: X can take on the possible or known values $\{v_1, v_2, v_3\}$
  - $\rightarrow$ Now encode these by separate inputs:

$$\begin{aligned} X = v_1 &\implies v_1 = 1, v_2 = 0, v_3 = 0 \\ X = v_2 &\implies v_1 = 0, v_2 = 1, v_3 = 0 \\ X = v_3 &\implies v_1 = 0, v_2 = 0, v_3 = 1 \end{aligned}$$

Next, we have the parameters influencing the **outputs**. NNs can be used for both classification and numeric prediction.

- For classification, the prediction of a class label given some input, we can either have one neuron to represent two classes (0 or 1), or one neuron per class, similar to One Hot encoding.
- Numeric prediction on the other hand has one contiuous-valued output neuron.

The next interesting parameter to investigate is the **weights**, which are the elements that are continuously updated to reduce the prediction error.

- For the initialization, we typically have random values assigned to each weight.
- Typical ranges for weights are: $w_{ij} \in [-1.0, 1.0]$ or $w_{ij} \in [-5.0, 5.0]$

For the **hidden layers** we first need to determine their amount and also the amount of neurons per hidden layer, as well as the activation functions.

- There is no clear rule as to the "best" number of hidden layers and neurons, but they may affect the accuracy of the resulting trained network.
- Network design is a trial-and-error process.

For the **training data** we have the following demands:

- The amount of training data is crucial for the correctness of the trained network.
- But, there is no robust way to indicate what the minimal size of the training set needs to be. As a rule of thumb, it goes:

$$\#\text{ training instances} \geq \underbrace{10}_{\text{some advocate 50}} \cdot \#\text{ weights in NN}$$

Important to mention: the parameters determining the hidden layers and the amount of training data are linked:

- A bigger NN allows to describe more sophisticated non-linear structures, BUT needs more training data
- Further, we have the constant battle between over- and under-fitting, as shown in a previous chpater (1.12).

7.5 Backpropagation

Now that we have the model and its basic structure and parameters, we'll look into how to train it.

Our **first step in training** is the random assignment of weights and then the measurement of the error.

$$Error\left(\vec{\mathbf{x}}, \vec{\mathbf{t}}, \vec{\mathbf{w}}\right) = \sum_{k=1}^{N} \frac{1}{2} \left(y_k - t_k\right)^2$$

With $\vec{\mathbf{x}}$: Input from training set
$\vec{\mathbf{t}}$: Labels (target values) — all vectors of size N
$\vec{\mathbf{y}}$: Calculated output with the help of $\vec{\mathbf{w}}$

Our **training goal** is to lower the error in each training episode by changing the weights. But in which direction should we change which weight?

- We have the same problem as before, and again we choose the steepest way down. The direction is known since we have the derivative of the function.
- But for our FNN, we have many weights. We need to pick a suitable step size, and we should do the update for all instances. For efficiency, this can be done in smaller batches.

So basically, we again apply gradient descent. But, when having thousands of neurons and millions of connections, the descent is performed in a space with millions or even billions of dimensions. To calculate how to nudge the weights, we therefore introduce the Neural Network Training Algorithm, better known as **Backpropagation** algorithm. It consists of the repetition of two passes:

Backpropagation

- First, we have the **Forward pass**:
  1. The network is activated on one example.
  2. Based on the calculated output, the *error of neurons of the output layer* is computed.

Forward pass

- Second, we have the **Backward pass**:
  1. Starting at the output layer, the *error is propagated backwards* through the network (layer by layer)
     - Recursively computing local deviation of error for each layer.
     - Use derivatives to preview the effect of a small change.
  2. Then the *weights are updated* (also for hidden layers).

Backward pass

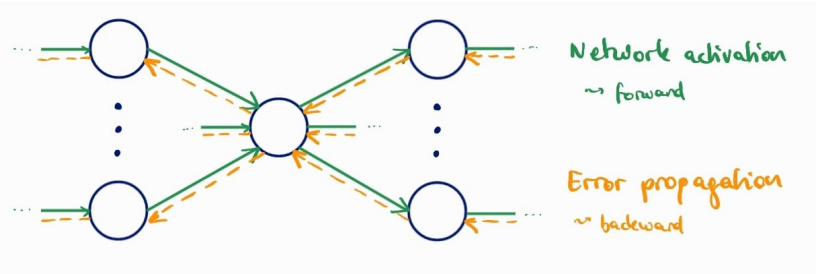


Figure 7.12: Backpropagation steps

To understand backpropagation, first, the concept of **derivatives** needs to be established:

$$\left.\frac{df(x)}{dx}\right|_{x=a} \text{ gives the slope of the curve } f \text{ at } x = a$$

Common Functions	$f(x)$	$\frac{d}{dx}f(x)$
Constant	c	0
Line	x	1
Line	ax	a
Square	x^2	$2x$
Square Root	$\sqrt{x}$	$\frac{1}{2}x^{-\frac{1}{2}}$
Exponential	e^x	e^x
Exponential	a^x	$\ln(a) \cdot a^x$
Logarithms	$\ln(x)$	$\frac{1}{x}$
Logarithms	$\log_a(x)$	$\frac{1}{x\ln(a)}$
Trigonometry	$\sin(x)$	$\cos(x)$
Trigonometry	$\cos(x)$	$-\sin(x)$
Trigonometry	$\tan(x)$	$\sec^2(x)$
Inverse Trigonometry	$\sin^{-1}(x)$	$\frac{1}{\sqrt{1-x^2}}$
Inverse Trigonometry	$\cos^{-1}(x)$	$-\frac{1}{\sqrt{1-x^2}}$
Inverse Trigonometry	$\tan^{-1}(x)$	$\frac{1}{1+x^2}$

Rules	$f(x)$	$\frac{d}{dx}f(x)$
Multiplication (const.)	cf	cf'
Power Rule	x^n	nx^{n-1}
Sum Rule	$f+g$	$f'+g'$
Difference Rule	$f-g$	$f'-g'$
Product Rule	fg	$fg'+f'g$
Quotient Rule	$\frac{f}{g}$	$\frac{f'g-g'f}{g^2}$
Reciprocal Rule	$\frac{1}{f}$	$-\frac{f'}{f^2}$
Chain Rule	$f \circ g$	$(f' \circ g) \cdot g'$
Chain Rule	$f(g(x))$	$f'(g(x)) \cdot g'(x)$
Chain Rule	$\frac{dy}{dx}$	$\frac{dy}{du} \cdot \frac{du}{dx}$

y_k, y_j, a_j, w_{ij} Consider the functions and variables we saw before for the two-layer FNN:

$$y_k(\vec{\mathbf{x}}, \vec{\mathbf{w}}) = f\left(\sum_{j=0}^{M} w_{ik}^{(2)} z_j\right) \quad \text{for } 1 \leq k \leq N$$

$$z_j = h(a_j) = \sigma(a_j) = \frac{1}{1+\exp(-a_j)}$$

$$a_j = \sum_{i=0}^{D} w_{ij}^{(1)} x_i$$

We'll calculate the weight update function for one single neuron j regarded in isolation.

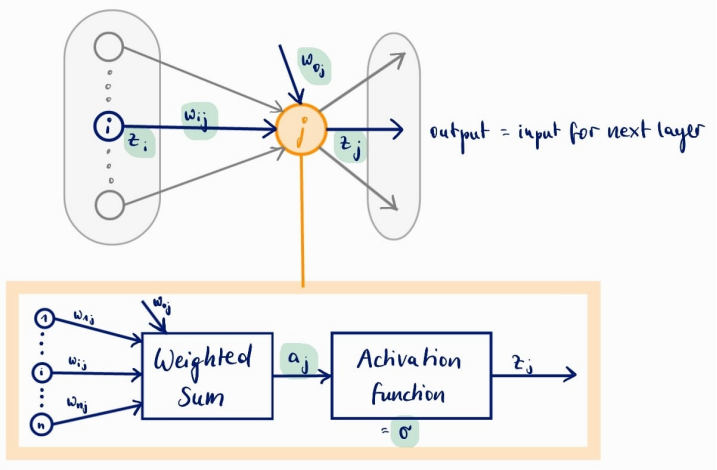


Figure 7.13: Isolated neuron components

Each neuron in the hidden layers as well as in the output layer takes its input, sums them up with weights, and then applies the activation function to it. Here, we chose the sigmoid (logistic) function. Generally, there are different activation functions h that can be used. But important for the backpropagation algorithm is that h is differentiable:

- Reason: we use the derivation of the error function, which is a variant of the gradient descent used for regression
- Figure 7.14 shows different activation functions, from which only the Sigmoid function is suitable for backpropagation

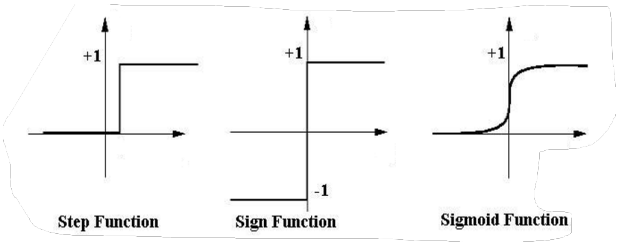


Figure 7.14: Activation functions (sigmoid)

What we now want to do with our isolated neuron j, whichever one this is, is to minimize the general error by nudging all its regarding weights w_{ij}, so we want to calculate $\frac{\partial Error}{\partial w_{ij}}$.

- Just quick note: for w_{0j} we have $z_0 = 1$ and hence $w_{0j} z_0 = 1$
- Our error is again defined as: $Error = \frac{1}{2} \sum_{\text{instances}} \sum_k (y_k - t_k)^2$ where the sum over instances is related to batch updating and epochs (in detail later)
- Important to mention about the indices: technically, we need to include the layer number (dropped here for simplicity)

Assume we notch the error definition a bit, and not regard the output of the whole NN but instead of a single layer. So we have:

$$Error = \frac{1}{2} \sum_k (z_k - t_k)^2$$

Further, we define $E_{ij} := -\frac{\partial Error}{\partial w_{ij}}$ to indicate the direction of the desired change for w_{ij}. With that, we define our update:

$$w_{ij}^{\text{new}} := w_{ij}^{\text{old}} + \partial w_{ij} = w_{ij}^{\text{old}} + \underbrace{l}_{\text{learning rate}} \cdot E_{ij}$$

- This introduces also a scaling parameter, precisely the learning rate l (similar to step size)
- The ∂w_{ij} aims to reduce the error

When we look at all our definitions, we can see that our w_{ij} only has effects "downstream" the network activation flow. We therefore can apply the chain rule. For that, we again introduce a new variable $E_j = -\frac{\partial Error}{\partial a_j}$.

$$\begin{aligned}
E_j = -\frac{\partial Error}{\partial a_j} &= -\frac{\partial Error}{\partial z_j}\frac{\partial z_j}{\partial a_j} = -\frac{\partial Error}{\partial z_j}\Big(\underbrace{\sigma(a_j)}_{=\frac{1}{1+e^{-a_j}}}\Big)' \\
&= -\frac{\partial Error}{\partial z_j}\cdot(-1)\frac{1}{(1+e^{-a_j})^2}\underbrace{e^{-a_j}\cdot(-1)}_{=(1+e^{-a_j})'} \\
&= -\frac{\partial Error}{\partial z_j}\underbrace{\frac{1}{1+e^{-a_j}}}_{=z_j}\underbrace{\frac{e^{-a_j}}{1+e^{-a_j}}}_{=\left(1-\frac{1}{1+e^{-a_j}}\right)} = -\frac{\partial Error}{\partial z_j}z_j(1-z_j)
\end{aligned}$$

$$\begin{aligned}
E_{ij} = -\frac{\partial Error}{\partial w_{ij}} &= -\frac{\partial Error}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} = E_j\frac{\partial a_j}{\partial w_{ij}} \\
&= E_j\frac{\partial(\sum_{i^*} w_{i^*j}z_{i^*})}{\partial w_{ij}} = E_j\left(\underbrace{\sum_{i^*\neq i}\frac{\partial w_{i^*j}z_{i^*}}{\partial w_{ij}}}_{=0} + \underbrace{\frac{\partial w_{ij}z_i}{\partial w_{ij}}}_{=z_i}\right) \\
&= E_j z_i
\end{aligned}$$

The last computation we need to do is $-\frac{\partial Error}{\partial z_j}$. For this case, we have two cases:

1. j is in the **output layer**

$$\begin{aligned}
\implies \quad & z_j = t_j \\
\implies \quad & -\frac{\partial Error}{\partial z_j} = -\frac{\partial\left(\frac{1}{2}\sum_{j^*}(z_{j^*}-t_{j^*})^2\right)}{\partial z_j} \\
& = -\frac{1}{2}\left(\underbrace{\sum_{j^*\neq j}(z_{j^*}-t_{j^*})^2}_{=0} + \underbrace{(z_j-t_j)^2}_{=z_j-t_j}\right) = t_j - z_j \\
\implies \quad & E_{ij} = E_j z_i = -\frac{\partial Error}{\partial z_j}z_j(1-z_j)z_i = (t_j-z_j)z_j(1-z_j)z_i \\
\implies \quad & \Delta w_{ij} = l\cdot z_i z_j(1-z_j)(t-z_j) \\
\text{And in particular} \quad & \Delta w_{0j} = l\cdot z_j(1-z_j)(t-z_j)
\end{aligned}$$

2. j is in a **hidden layer**, which means changes in z_j impact all a_k values in the next layer proportional to the connection weight.

$$\implies \quad -\frac{\partial Error}{\partial z_j} = -\sum_k \underbrace{\frac{\partial Error}{\partial a_k}}_{=-E_k} \frac{\partial a_k}{\partial z_j}$$

$$= -\sum_k -E_k \underbrace{\frac{\partial \sum_{j^*} w_{j^*k} z_{j^*}}{\partial z_j}}_{=\frac{\partial w_{jk} z_j}{\partial z_j} = w_{jk}} = \sum_k w_{jk} E_k$$

$$\implies \quad E_j = z_j(1-z_j)\sum_k w_{jk} E_k$$

$$\implies \quad E_{ij} = z_i z_j (1-z_j)\sum_k w_{jk} E_k$$

$$\implies \quad \Delta w_{ij} = l \cdot z_i z_j (1-z_j)\sum_k w_{jk} E_k$$

And in particular $\quad \Delta w_{0j} = l \cdot z_j (1-z_j)\sum_k w_{jk} E_k$

Summarized, we have explicit expressions for our updates:

Weight update, E_{ij}, E_j, Δw_{ij}

$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + \Delta w_{ij}$ — updated weight of connection from neuron i to neuron j based on some training instance

$\Delta w_{ij} = l E_{ij} = l E_j z_i$

$\Delta w_j = l E_j$

$E_j = z_j(1-z_j)(t-z_j)$ — **Case 1:** neuron j is in the output layer

$E_j = z_j(1-z_j)\sum_k w_{jk} E_k$ — **Case 2:** neuron j is in a hidden layer

With scaling parameter l as the learning rate

Next, we're gonna investigate the **learning rate**, which is usually a constant between 0.0 and 1.0.

Learning rate l

- As a rule of thumb: $l = \frac{1}{r}$ with r being the "round", so the number of iterations
- It can also help to start with bigger steps and end with smaller ones to help a quicker convergence while avoiding overshooting the target
- The choice of l is similar to the choice of the step size in regression and SVMs
- Typical problem:

  l too small : learning occurs at very slow pace

  l too large : oscillation between inadequate solutions can occur

An important thing to consider for backpropagation is **when to update**:

- In the case of **instance updating** we update the NN for each instance (sample) individually.

  Instance updating
- Contrary, when we apply **batch updating** the NN is updated for a subset (or all) of all training instances in one update action. So we have an error calculation based on the same weights for multiple instances

  Batch updating

- Best practice is using **mini batches**, where the size depends on the problem.

Epoch

- Another important term is **epoch**, which describes one complete consideration of all the training data.

Termination

Finally, there's also the consideration for the **termination** of the backpropagation algorithm. There are multiple termination criteria indicating to stop training.

- All Δw_{ij} in the previous epoch are small (below some specified threshold)
- Percentage of misclassified inputs in previous epoch is small
- Pre-specified number of epochs has expired
- In practice: Several hundreds of thousands of epochs may be required before weights are converging

7.6 Beyond Basic FNNs

The general application area for NNs is supervised learning. This has been the focus and most dominant usage.

- This creates the requirement of labeled data as input, potentially a large amount of those
- There is a huge variety of network types, as we saw before, also optimized for different input types:
  - CNNs use convolution to better deal with images
  - RNNs and LSTM (Recurrent NN, Long Short-Term Memory) are tailored for temporal or sequential data
- Generally, NNs require lots of engineering and trial and error (no magic bullet, no free lunch)

Beyond supervised learning, there are also **unsupervised learning** applications, where we don't use classes or similar things as our target.

Autoencoder

For example, in the case of **autoencoders**, the "label" is produced from the input such that the input is tried to be reproduced. The error is the difference between input and calculated output. The interesting part is the significantly more dense representation of the input data captured by the middle layer, as can be seen in 7.15.

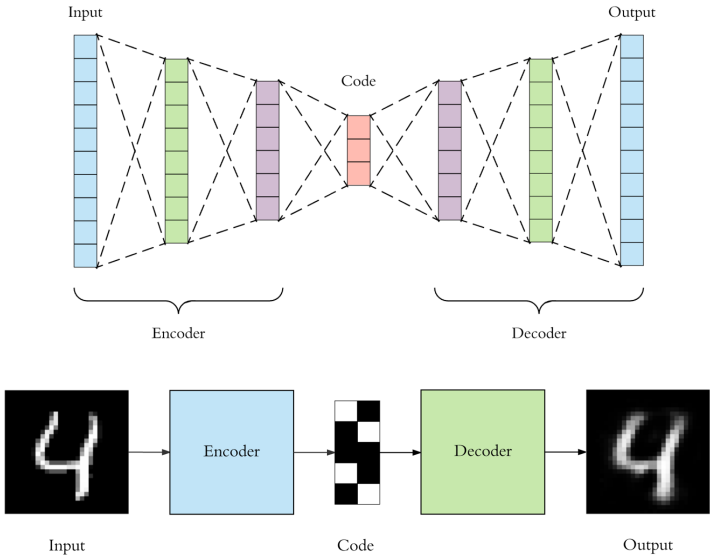


Figure 7.15: Unsupervised learning: autoencoder

Another application is **GANs** (generative adversarial networks). Here, we have two networks: Generative Adversarial Networks

- One network is the generator that tries to create "fake data" (e.g. fake art, fake people)
- The other is the discriminator that tries to distinguish "real" from "fake data"
- This means, they have opposing goals

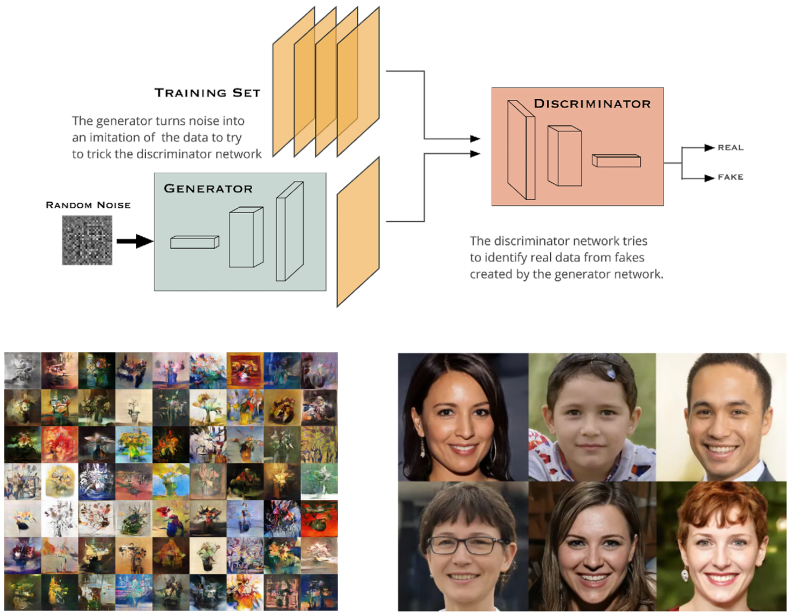
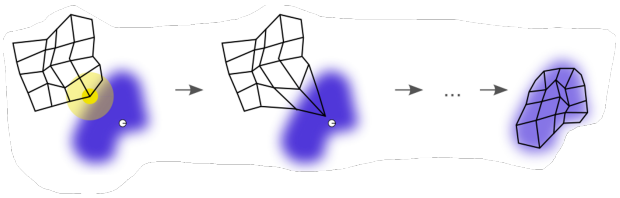


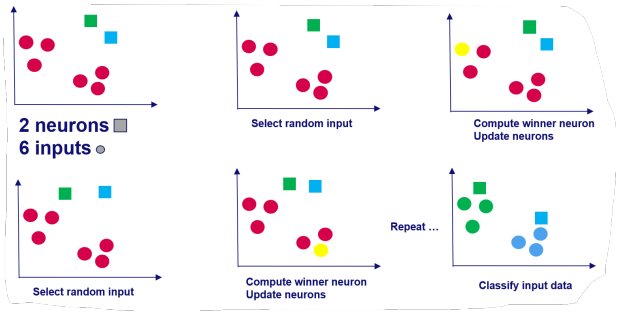
Figure 7.16: Unsupervised learning: GANs

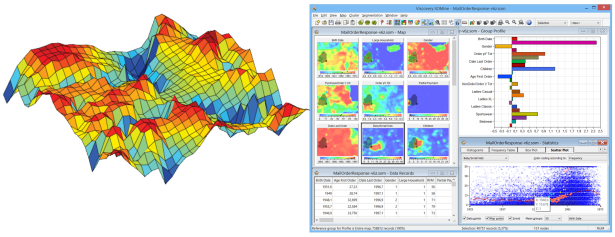
A final unsupervised application is **self-organizing maps** which is a directly unsuper- Self-organizing maps

vised neural network.

- We have a distribution of the training data (blue blob)
- In the beginning, the SOM nodes are arbitrarily positioned in the data space
- Then the following process is iterated until all input data has been seen:
  - Randomly select a training data point as input (white point)
  - The node nearest to the training node is selected as winner (yellow point, with neighbors in yellow area)
  - It is then moved toward the training data, just as the neighbors on the grid (just to a lesser extent), so update the neurons
- Then the input can be classified
  - The inputs are connected to the neurons with weights
  - The neurons are also related to each other







Natural grouping of instances in the two-dimensional grid (each cell colored using any of the features)

Figure 7.17: Unsupervised learning: SOMs