Introduction to Data Science

Release 0.1

Steffen Herbold

Jul 14, 2020

CONTENTS:

PREFACE

Welcome to the online book *Introduction to Data Science*. This book is created to provide a great resource for asynchronous online learning to deal with the current pandemic, where physical lectures are not possible and not all participants may be able to attend lectures, e.g., due to health issues or just because you have to care of a kid.

However, this online book is more than just a short term fix due to the pandemic. This book is an interactive guide for getting started with data science. All code that is used, e.g., to perform analysis or to create visualizations is included and can be re-used by anyone.

The book is provided both online and as a PDF for printing.

Target audience

The primary audience are students, who visit my courses at the university. These students usually have (some) computer science background, but there are also always interested students who have only little prior knowledge of computer science. The book is written such that anyone interested in data science can follow *most* of the book, but maybe not every detail.

Prerequisites

In general, this book is designed to be relatively easy to follow for students at the university level who have some affinity for working with data and ideally also for programming and mathematics. However, without at least some knowledge about mathematics and computer science, a complete understanding of the materials will not be possible, even though you should be able to understand most concepts.

The following skills are required for a complete understanding of this script:

- Basic programming skills, ideally some Python knowledge. All contents of the script, with the exception of the programming examples, should be understandable without programming knowledge.
- Mathematical notations commonly used in higher mathematics. Without this knowledge, you may not understand how all models for data analysis are working. Here are some example of simple formula that you should be able to read:
- $\sum_{i=1}^{n} x_i$ (sum of the values $x_1, \ldots, x_n$)
- $\Pi_{i=1}^{n} x_i$ (product of the values $x_1, \ldots, x_n$)
- $x = (x_1, ..., x_d) \in \mathbb{R}^d$ (x is a d-dimensional real valued vector)

- Stochastics, especially random variables and their distributions, e.g. normal/gaussian distribution, uniform distribution, exponential distribution, and binomial distribution. Without this knowledge you will have trouble understanding certain problems with data and models.

Installation of Required Software

This online book is written using Juypter Notebooks. You can download the Jupyter Notebooks of each chapter using the download link at the top of the pages. A PDF with the content of all chapters is also planned, but not yet available.

If you want to run the Jupyter Notebooks yourself, i.e., execute the code examples provided, you must install the required software. These are

- Python 3.6 or newer.
- Jupyter Notebook (tested with version 5.7.8, but should likely also work with all newer and most older versions).
- The python libraries numpy, scipy, scikit-learn (at least 0.22), pandas, matplotlib, seaborn, mlxtend, nltk, and wordcloud.

We recommend to setup a local environment as follows: - Install Python - Install Jupyter Notebook. We recommend installation with pip, because you get this delivered with Python. More experienced users can also use conda. - Install the required libraries with pip (`pip install numpy scipy scikit-learn pandas matplotlib mlxtend seaborn nltk wordcloud`). This may require sudo/admin rights. You can also install this in your user space if you add `--user` to the command. The creation of the neural network figure in Chapter 7 also requires the package `graphviz`, which you have to install both with pip, as well as within your operating system.

You should now be able to fire up your installation of Jupyter with the command `jupyter notebook`. Jupyter Notebook runs in the browser and initially shows the home folder in your local file system. You can navigate to the destination where you downloaded the notebooks and open them. You could also upload them to the currently open folder using the browser.

State of the Book

This online book is still a work-in-progress and it will take some time before the book is finished. This section summarizes the current state.

- Chapter 1: Finished.
- Chapter 2: Finished.
- Chapter 3: Finished.
- Chapter 4: Finished.
- Chapter 5: Finished.
- Chapter 6: Finished.
- Chapter 7: Finished.
- Chapter 8: Finished.
- Chapter 9: Finished.
- Chapter 10: Finished.
- Chapter 11: Finished.
- Chapter 12: Draft finished, requires proof reading.

- Exercises: Finished.
- Solutions: Finished.
- Appendix: Finished.

If you find problems (e.g., spelling errors in chapters that are already proof read, blunders on my part like wrong formulas, formatting problems on the Website or in the PDF), please open an issue on GitHub.

CHAPTER

ONE

BIG DATA AND DATA SCIENCE

1.1 Introduction to Big Data

The term *big data* has been around for quite some time and the initial hype associated with the topic is already long over and replaced by different topics, like the *internet of things* and *artificial intelligence*, especially *deep neural networks*. However, big data is closely related to these topics and can be, to some degree, seen as an enabler and important related technology.

Unfortunately, many people still do not understand what makes data *big*, what is special about big data, and the implications of having problems that either produce or require big data. These misunderstandings are mainly because we instinctively assign a meaning to the term big in the sense of large. Consequently, the naive definition of big data would be that this is just a large amount of data. However, only the amount of data is not sufficient to make data into big data. Consider, for example, the backups that modern compute centers store on high volume storages, that often have a high latency. This is certainly a large amount of data, but also certainly not big data. Moreover, defining big data just using the size would be impractical, because we would have to change the definition repeatedly, because our storage, computational, and memory capacities are constantly increasing.

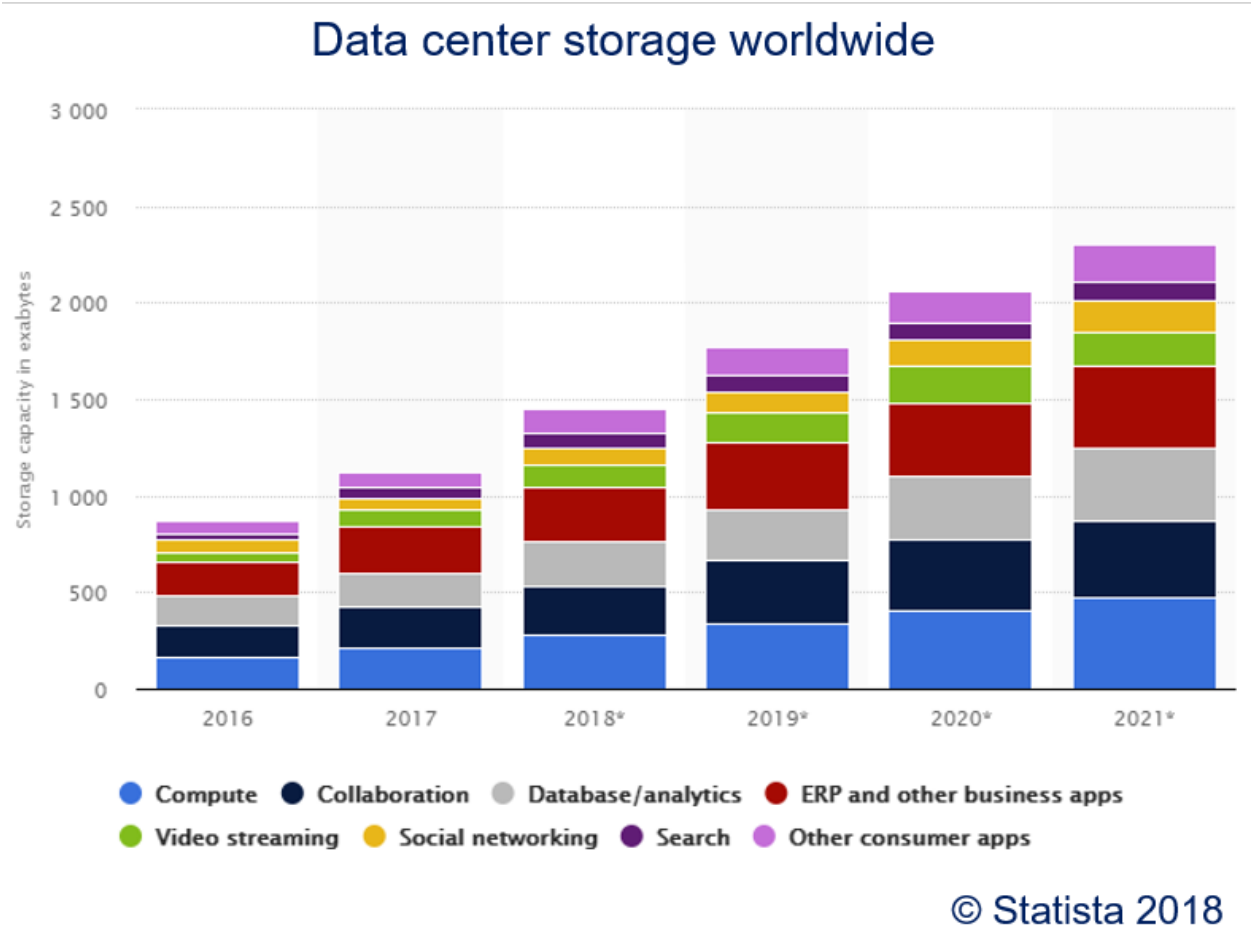
A better and well accepted definition of big data is based on the *three Vs* [Gartners IT Lexicon]:

> **Definition of big data:**
>
> Big data is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation.

Let us dissect this definition and consider the individual parts to understand why big data is more than just size.

1.1.1 Volume

The volume of the data is an important factor. However, as discussed above, there is no exact number that makes data big. In the year 2006, when Google published the landmark Map/Reduce paper, one Terabyte was a huge amount of data. In the year 2020, this is the storage of my laptop. A simplistic definition is that big data does not fit in the memory. A better guideline is that big data is too large to be copied around (very often), especially over the network. Due to this, big data is sometimes not even copied from the network but with the Sneaker Net, for example, for the creation of the first image of a black hole.

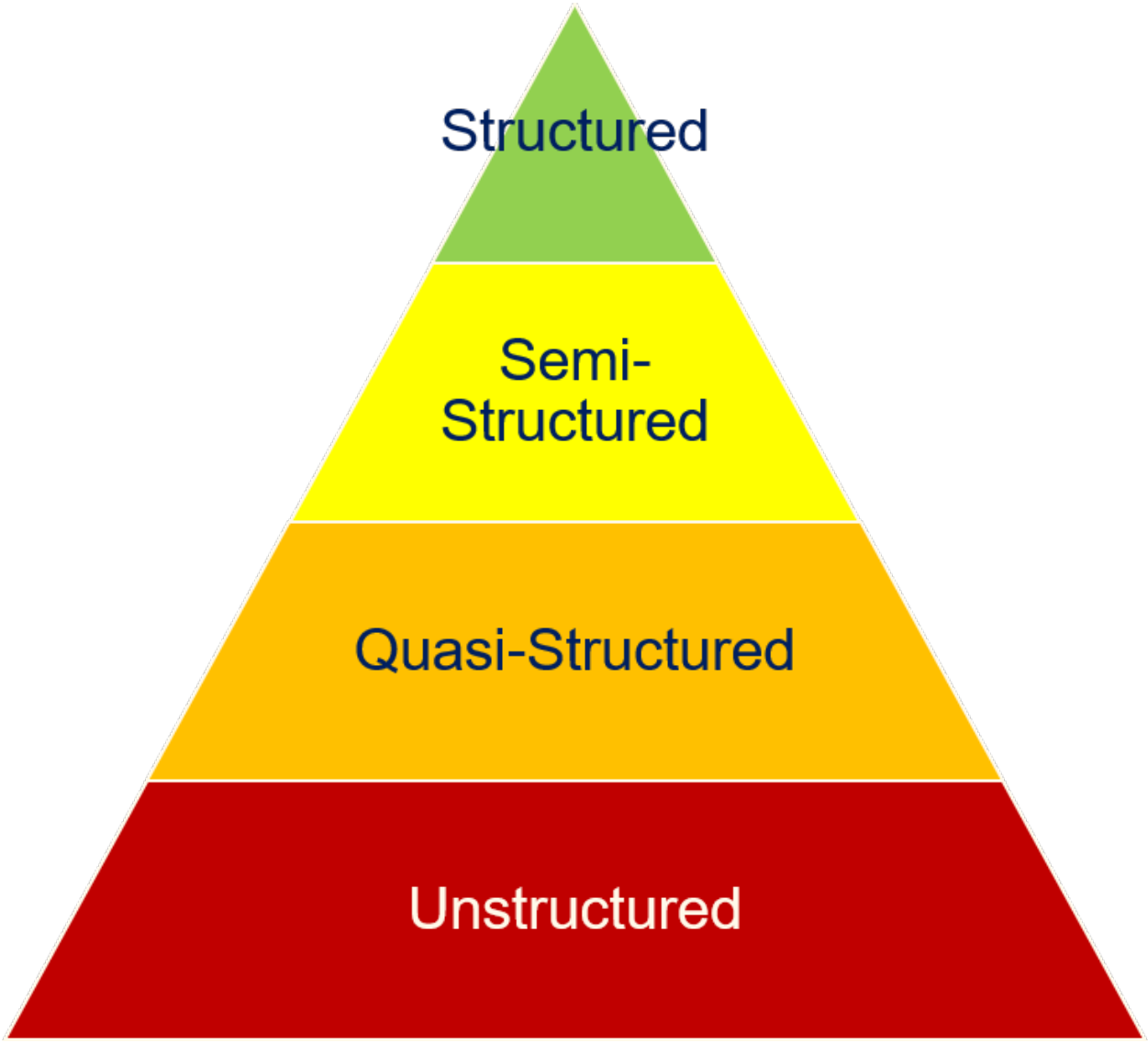


1.1.2 Velocity

The velocity deals with the speed at which new data is created, processed, and/or evaluated. Examples for high velocity data are the different sensors in self-driving cars (LIDAR, Cameras, ...). Such data can become large really fast. For example, Waymo released compressed data from about eleven hours of driving which is two Terabytes of data [Waymo Open]. Data that is produced at such high speeds is also called *stream data*. Often, streaming data must be processed in (nearly) real-time. Thus, there are not only requirements on being able to receive large amounts of data that are created rapidly, but also to process and evaluate this data in a timely manner. Consequently, one might say that there is an inverse correlation between the velocity and the volume when it comes to deciding if data is big: the higher the velocity, the smaller the volume that is required to make data big. The reason is quite simple: processing one Gigabyte in one hour is simple, processing one Gigabyte per second is a different problem.

1.1.3 Variety

The variety is the third aspect of big data. Today, the analysis of images, videos and text has become a relatively normal application. However, this was not the case when the term big data was initially coined. Then, most data was held in data warehouses with defined structures, for example, relational databases. The data were mostly numeric or in fixed categories. This began to change when the Internet started to grow into the almost all-encompassing web of knowledge and content we have today. It is no coincidence that Google was at the forefront of the development of big data technologies, because the initial driver was indexing the Internet to enable efficient queries for finding content. This meant that unstructured data, like websites must be indexed and information retrieval algorithms must be executed against this data. The amount of unstructured data is vastly more than that of structured data. Typically, the relation between data structuring and the volume of data is depicted as a pyramid.



At the top of the pyramid is the *structured data*, e.g., tables, comma separated value files and similar. Often, this data can be directly analyzed and pre-processing is only required for data cleaning, e.g., the detection of invalid data or outliers.

Next, we have *semi-structured data*, e.g., XML or JSON files. The main difference between structured and semi-structured data is that semi-structured data formats are often more flexible. For example, each row in the table of a relational database must have values for exactly the same columns. With XML and JSON, the fields are usually

similar, but may have structural differences, e.g., due to optional fields.

The first two layers of the pyramid are defined data formats for which there are usually query languages and/or libraries for the extraction of information. This is not the case on the bottom layers. *Quasi-structured* data has a fixed structure, but not in a convenient and easily accessible data format. For example, consider the output of the `ls -l` command.

```
%ls -l
```

```
total 36
-rw-rw-rw- 1 sherbold sherbold 24996 Mar 26 11:04 01_Introduction.ipynb
-rw-rw-rw- 1 sherbold sherbold  6302 Mar 26 13:32 02_Process.ipynb
drwxrwxrwx 1 sherbold sherbold   512 Mar 26 11:21 images/
```

There certainly is a discernable structure in the output, i.e., *most* lines contain a summary of the user rights, followed by the number of links, the user and group who own the file, the size, the data of the last change, and the name. This structure can be exploited to define a parser for the data, for example, using regular expressions. Thus, we are able to impose a structure on the quasi-structured data, by defining the meaning of the structure on our own and writing our own parsers for the data. A potential problem with quasi-structured data is that these data formats are often not very reliable and may change. For example, `ls` could separate fields with tabulators instead of spaces, which would break most parsers. There is no protection against such changes, which makes such parsers fragile and may mean that significant effort must be invested for the maintenance of parsers for quasi-structured data in production environments.

On the bottom layer of the pyramid is the *unstructured data*, which is the vast majority of available data, e.g., images, videos, and text. The challenge of unstructured data is that a structure must be imposed for analyzing the data. How this is done depends both on the data and the application. Moreover, there are often mixed formats with unstructured data. Just consider this script. We have a mixture of natural language text, images, markdown information that specifies special features of the text (headlines, listing), and even source code. How the structure is imposed depends both on the data, as well as the application.

1.1.4 Innovative forms of information processing

While the three Vs are usually considered as the major aspects of the definition of big data, the other parts of the definition are also important to understand why big data is not just more data, that may be generated rapidly and different formats. The next part of the definition states that *innovative forms of information processing* are required. This means you cannot just use a normal workstation or even a traditional batch system, where you have many computing resources to which a shared storage is attached via the network. Instead, *data locality* becomes an issue, i.e., preventing copies of the data due to the volume. This requires different infrastructures in which computational power and storage is combined. When big data was a new concept, such technologies basically did not exist. Nowadays, there are many ways to implement such infrastructures, e.g., with Hadoop, Spark, Kafka, Cassandra, HBase, and many others.

1.1.5 Insights, decision making, and process automation

The final part of the definition means that having large amounts of data is not yet big data. Data can only become big data, if it is actually used, e.g., to generate insights, guide decision making, or even automated parts of a business process. This aspect is so important, that there are also definitions of big data in which there is an additional V for *value*.

1.1.6 More Vs

We use a definition with three Vs for big data. Using words that start with V is so popular for big data, that there were multiple suggestions to extend the definition with additional Vs, with up to 42 Vs. Obviously, this is too much and was created with the goal to show that more Vs do not mean that we have a better definition for big data. Regardless, for up to Ten Vs, there are more serious definitions. We already met one the additional Vs, the value which is just called differently in our definition. *Veracity* is another important V that deals with the quality of the data. The more data you have, the harder it is to ensure that the data is reliable and the results can actually be reproduced. This is especially important if the data source changes often, e.g., if news outlets or social network data are analyzed. Volume, velocity, variety, veracity and value are the five V definition of Big Data, which is also popular. We do not cover any of the other Vs here.

1.2 Data Science and Business Intelligence

Data science is a relatively new term for which no agreed upon definition has emerged yet. The reason for this is likely two-fold. On the one hand, the term is very generic, i.e., every use of anything related to data that is remotely scientific can be coined as data science. On the other hand, there is a major hype surrounding the term, which means that companies, consulting firms, funding agencies, and public institutions want to advertise with their use and support of data science.

Due to this, we also do not try do find a good and concise definition for data science. Instead, we look at examples for things that fall under the term data scientist and try do understand the differences to a term that was also popular in the industry a couple of years ago, i.e., business intelligence.

1.2.1 What is Data Science?

Data science brings together mathematics, statistics, and computer science with the goal to generate insights and applications from data.

Mathematics plays a foundational role in how we work with data, because a common goal of many data science projects is to find a mathematical description for a certain aspect related of the data. Thus, data science is ultimately about finding mathematical models. However, the impact on mathematics goes beyond just being a "description language" for models about data. Various fields of mathematics are integral parts of the methods we use to determine models, for example, the following.

- *Optimization* deals with the question how optimal solutions for a target function can be found in a space of possible solutions described by constraints. This is often used to optimize the models we derive from data.
- *Stochastics* is used to describe the behavior of random events through random variables and stochastic process. These are the foundation for the theory of machine learning as well as for many applications.
- *Computational geometry* is required for analyzing data that is spatially distributed, e.g., geographically, astronomically, or on the 3D space in front of a car.
- *Scientific computing* is also related to data science, because more and more applications emerge where machine learning and classical scientific computing are used together.

Statistics deals with the analysis of samples of data through the inference of probability distributions that describe the data, time series analysis, and the definition of statistical tests that evaluate if assumptions on the data likely hold. Concrete aspects from statistics that are relevant for data science are, for example, the following.

- *Linear models* are a versatile means to fit linear descriptions to data for the analysis and may also be used for forecasting future values.
- *Inference* is a similar method for describing data, but mostly through probability distributions instead of linear models.

- *Statistical tests* are an important part in the toolbox of any scientists and can be used to determine how well models work, especially if it is likely that observed effects are only random.
- *Time series analysis* exploits structural patterns in temporal data to analyze the internal structure of data over time and may also be used to forecast future values.

The mathematics and statistics would not be actionable on data without computer science. Additionally, theoretical computer science is also part of the foundations of data science. Examples for concepts from computer science that are relevant for data science are the following.

- *Data structures and algorithms* are the foundation of any efficiently implemented algorithm and the understanding of data structures like trees, hash maps, and lists as well as the run time complexity of algorithms enables the understanding and implementation of efficient data science approaches.
- *Information theory* covers the concepts of entropy and mutual information which are important for many algorithms that are used for data analysis.
- *Databases* are the foundation of efficient storage, access, and nowadays even computation with data and SQL is an invaluable skill for any data scientists, that can often even be used with NoSQL databases.
- *Parallel and distributed computing* is a pre-requisite for any Big Data analysis, the scaling of problems to large groups of users, and the efficient implementation of run time extensive algorithms.
- *Artificial intelligence* deals with logical systems and reasoning that can also be applied in modern data science applications. Please note that we explicitly distinguish between artificial intelligence and machine learning in this script. We use the term artificial intelligence for applications like Deep Blue, the rule-based chess system that was the first computer to beat Gary Kasparow in Chess.
- *Software Engineering* is important for any data science approach that should be implemented in a production system, but also for the general management of data science projects.

Finally, there is *machine learning*, which is parts mathematics, parts statistics, and parts computer science, depending on which approaches for learning you want to use. Machine learning tries to infer knowledge from data and generalize this knowledge to other contexts, e.g., through neural networks, support vector machines, decision trees, or similar algorithms.

1.2.2 Examples for Applications

The field of data science is diverse and has many applications in research, industry, and society. Here are six short examples.

- *Alpha Go* is an example for an intelligent self-learning system. A couple of years ago, Alpha Go surprised the world because it came from seemingly nowhere and beat one of the best players of the game of Go. This was surprising, because prior to Alpha Go, computers were on the level of amateurs when it came to go and far away from even being a challenge for professional players. Alpha Go combined classical rule-based artificial intelligence with a self-learning recurrent neural network, to achieve this.
- *Robotics* relies on machine learning to improve how robots move. Boston dynamics is famous for teaching robots a sense of balance by pushing the robots. The robots *learn* how to avoid falling down over time, the same way toddlers learn this.
- *Marketing* and more specifically targeted advertisements in the Internet are a billion-dollar market based on learning which ads are most relevant for users based on their browsing behavior.
- *Medicine* relies more and more on data driven decision support. IBM Watson, who was initially famous because this was the first artificial intelligence that could beat humans in jeopardy, is now being used to help make decisions about cancer treatments. (Although this is not working as well as hoped for.)
- *Autonomous driving* relies on machine learning for different tasks, most importantly the recognition of objects like other cars, bikes, and pedestrians.

1.2.3 Differences to Business Intelligence

In the industry, business intelligence is a related ancestor of data science that has been in use for years. Gartner defines Business Intelligence as "best practices that enable access to and analysis of information to improve and optimize decisions and performance." Consequently, for many organizations data science is just a rebranding of business intelligence. However, a closer look at typical data science applications and business intelligence applications reveals the differences between the terms. The following table compares the typical techniques, data types, and common questions of business intelligence and data science.

	Business Intelligence	Data Science
Techniques	Dashboards, queries, alerts	Optimization, predictive modelling, forecasting
Data Types	Structured, data warehouses	Any kind, often unstructured
Common Questions	What happened? How much did? When did?	What if? What will? How can we?

As can be seen, business intelligence is focused on the analysis and reporting of the past. Data is typically stored in databases, structured and ready to be analyzed. Data science is more or less a superset. Everything from business intelligence may also be coined as data science, however, data science goes beyond that by considering the future. Thus, data science tries to generalize from the data such that forecasts and predictions are possible, which means more complex questions can be answered, e.g., how different scenarios will play out. This allows deeper insights than business intelligence.

1.3 The Skills of Data Scientists

Data scientists are not computer scientists, mathematicians, statisticians, or domain experts. Instead, the perfect data scientist is a combination of all of that.

- Good mathematics skills, especially about optimization and stochastics.
- Statistician with knowledge about regression, statistical tests, and similar techniques.
- Computer science skills, including programming, databases, algorithms, data structures, parallel computing, and ideally also Big data infrastructures.
- Strong knowledge in the intersection of the fields, especially machine learning.
- Enough domain knowledge to understand the data, the questions that must be answered, and how the questions can be answered with the available data.

Soft skills are also important for data scientists. Team work is often required, as data scientists often work at the intersection between domain experts on the one hand, and technical staff on the other hand. The domain experts teach the data scientists about data, the questions that should be answered, and how the outcome of projects should affect future research and/or business processes. The technical staff often takes over at some point when (and if!) projects are operationalized.

Moreover, the data *scientist* should be skeptical and follow the scientific method. This is especially important when dealing with data, to rule out that effects are purely random.

Because this is a very diverse and complex skill set, the proportion of people who can do all of the above is relatively small. Microsoft Research performed a survey with Microsoft employees to determine which tasks related to data science work on. They found that there are nine different types of data scientists.

- *Polymaths* are general purpose data scientists who fit the complete profile described above, i.e., those who can really do it all, from the underlying mathematics to the deployment of big data infrastructures.

- *Data Evangelists* perform data analysis and actively push for the adoption of data driven methods as well as acting on the gained insights.
- *Data Preparers* query existing data platforms and prepare the data for the analysis.
- *Data Shapers* also work on the preparation of the data but also analyze the data.
- *Data Analyzers* use already prepared data and analyze the data to generate insights.
- *Platform Builders* collect data as well as create and administrate platforms both for the collection and analysis of the data.
- *Moonlighters 50% / Moonlighters 20%* are part-time data scientists, that contribute to data science projects but only in a fraction of their overall work.
- *Insight Actors* use the outcome from data science projects and act on the insights.

CHAPTER

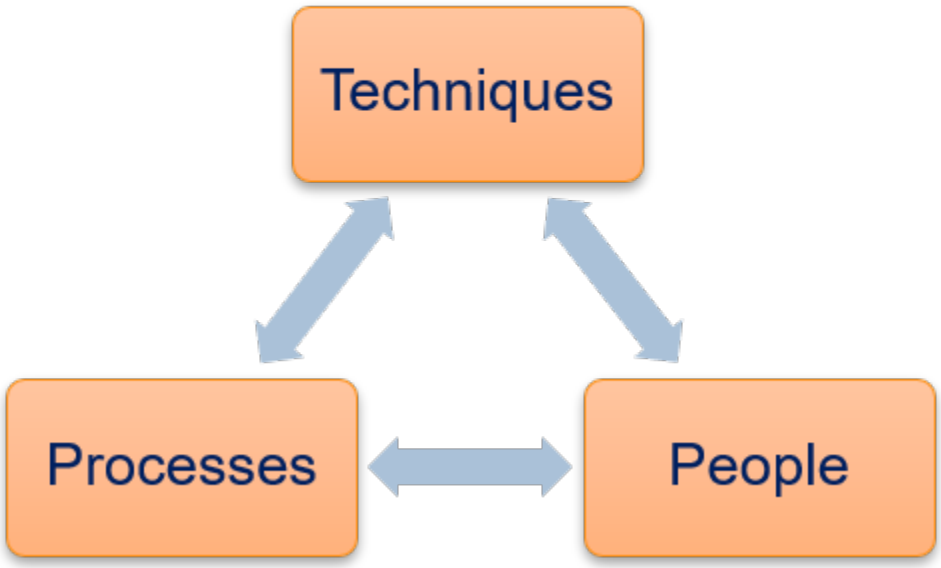
TWO

THE PROCESS OF DATA SCIENCE PROJECTS

2.1 Generic Process Model

2.1.1 Processes

Processes are at the core of any activity, even though we are often not even aware of that. Activities are executed by *people* who apply *techniques*. The *process* guides and organizes the activities of the people and describes the techniques that are used.



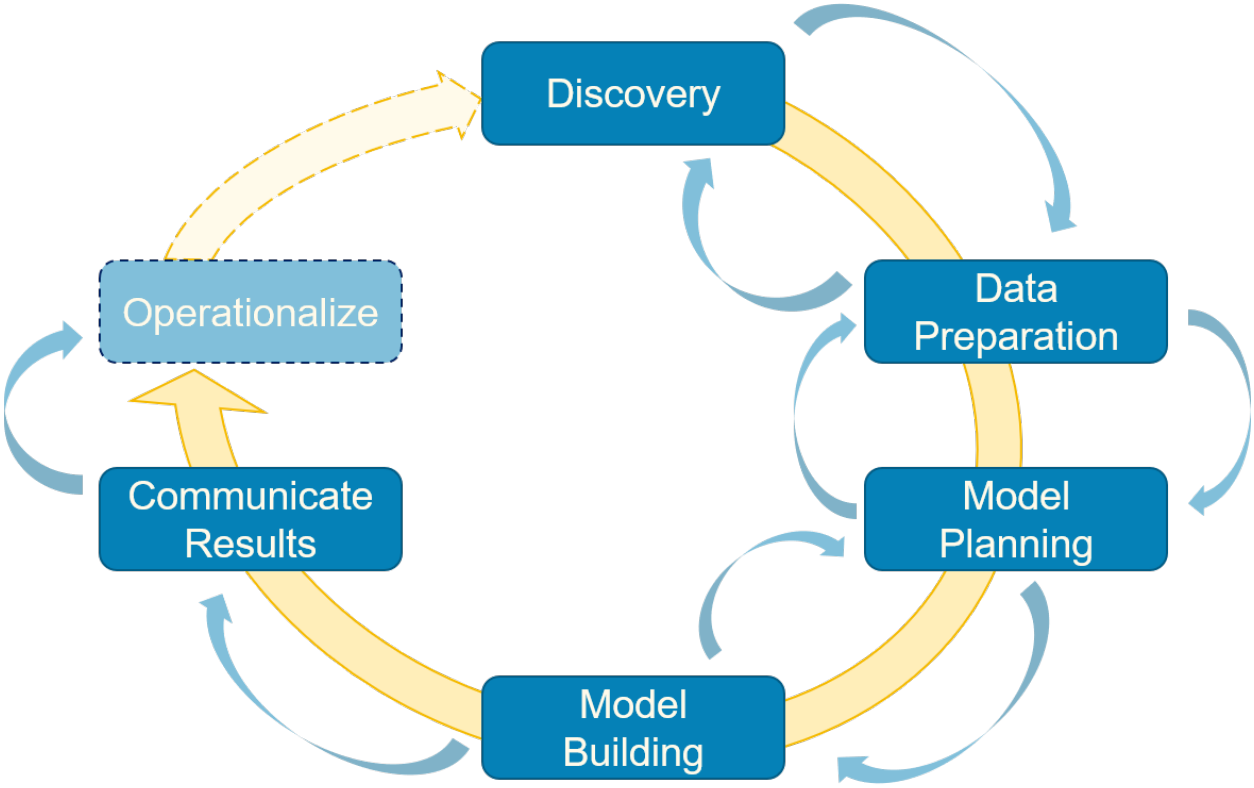
The goal of a good process model is to support the people, e.g., by ensuring that important activities are not forgotten and the recommendation of suitable tools for the solution of problems. In general, process models describe *best practices* that should be applied due to their past success. Through this, the reliance on the existing knowledge and skills of the people should be reduced with the aim to also reduce the risk of project failures. Processes must be supported by the people that use them. If the people do not accept a process, this can hinder productivity and increase the risk of project failures. To ensure that this is not the case, processes should have a measurable positive effect.

A good process requires that the people receive the necessary training for the techniques that should be applied. Moreover, the techniques must be suited for the project. In general, there is no "one fits all" process, because different

aspects influence the choice of processes and techniques, e.g., the project size, whether there are safety-critical aspects, and prior knowledge of the people.

2.1.2 Overview of the Generic Process of Data Science Projects

Our generic process does not prescribe specific techniques, but only the general phases of a project, i.e., a rough sketch of the required activities. For data science projects, there are six generic phases as is shown below.



The process is iterative, i.e., there may be multiple repetitions of the phases in a project. Within one iteration, it is only possible to jump back to prior phases, until the results are communicated. The reason for this is obvious. At this point you decide that these are your results and you communicate them to a broader audience, e.g., the upper management, your customers, other researchers in form of a publication or the submission of your thesis. In the following, we consider each phase in detail.

Discovery

The discovery is the initial phase of the project. The main goal of the discovery is to understand the domain, objectives, data, and to decide if the project has sufficient resources to go forward. To achieve this, many activities must be performed.

The data scientists must *acquire the required knowledge about the domain*, in which the project is conducted. This means that the data scientists must understand the use case associated with the project. Domain experts often collaborate with the data scientists and provide the necessary explanations, e.g., in form of documents or through interviews and workshops. As part of this, the data scientists also gain the required knowledge about the data, i.e., a first and vital understanding about the available information assets that will be used for the analysis in the project. The gained knowledge helps the data scientists to understand the project, as well as with the interpretation of the project results.

Part of the learning about the domain should also be a *consideration of the past*. For academic projects, this is standard practice, as the related work must always be reviewed and considered carefully for any project. However, this is also valuable for work in the industry. Possibly, similar projects were attempted in the past. If this is the case, the results - both positive and negative - from the past projects are invaluable, as they help to avoid similar mistakes and provide guidance about working solutions. Within the bounds of the copyright and patent law, an analysis of the solutions of competitors may also help to better understand the problem as well as potential solutions.

Once the data scientist gained sufficient knowledge about the project, she can start to *frame the problem*. This means that the problem that shall be solved is framed as a data analysis problem. This is different from the goal of the project, which is usually a general business or research objective. The previously acquired domain knowledge is invaluable for this, as the data scientist must understand why the problem is important for the customer in order to frame it correctly. Typical questions that the data scientist must answer for this are, for example, who the stakeholders are and what their interest in the project is. The data scientists learn the current problems (pain points) as well as the goals of the stakeholders from this analysis. Based on this assessment, the objectives of the project can be clearly formulated, as well as how the success of the project will be determined. However, data scientists should not only think about the success, but also about risks that may lead to project failure by missing the objectives.

As part of all of the above, the data scientists learn about the data that may be used for the project. The data may already be readily available, e.g., in a data warehouse. However, it is also possible that data must be collected. In either case, the data scientist must get initial knowledge about the scope and structure of the data and gain a high-level understanding of the available information. Otherwise, a subsequent assessment of the required resources would not be possible.

The *science* part of data science should also not be neglected during the discovery. This means that data analysis should not be purely exploratory, but that clear expectations in form of hypotheses should be formulated that can be tested. Otherwise, there is a high chance that results of the projects do not generalize. Moreover, these hypotheses guide the subsequent phases of the project, especially the model planning and model building. These hypotheses should be discussed with domain experts.

Once the project is completely understood, the final step of the discovery is to decide whether to go forward with the project or not. This assessment should be done based on the risk assessment, as well as on whether the available resources are sufficient for execution of the project. At least the following resources should be considered:

- Technological resources, including resources for the data storage, computational resources, and possibly also whether the required software licenses are available or can be bought.
- The required data, i.e., if the required data is available or can be reasonable collected within the scope of the project. This assessment should look at two dimensions, i.e., the number of data points of the data that must be sufficient to achieve the objectives, as well as the information available for each data point is sufficient. Please note that the collection of additional data should be considered during the assessment of the project risks.
- The available time, both in terms of calendar time and person months. Calendar time is the duration of the project. For projects with a calendar time less than one year, the months in which the project is executed should be considered, as holiday seasons may significantly reduce the availability of personal. This further depends on the geographic distribution of the project team, as different times are critical in different countries (e.g., Lunar new year, Christmas holidays, or more or less the complete August in some countries). Person months are an estimate for the effort that developers and data scientists spent on the project. However, we note that two persons working for one year are usually not twice as productive in a project as a single person, which should be taken into account. This phenomenon is well-known and described in The Mythical Man-Month.
- Human resources, i.e., the concrete personal that should work on the project, including whether the skill set of the personal matches the requirements of the projects.

Projects should only be started if all required resources are available.

Example

Your customer is the owner of a Web shop that sells clothing. They want to increase their sales through cross-selling and ask you to design a solution for this based on data about their past sales. As part of the

discovery, you may do the following:

- You interview the customers to better understand if they already have an idea how they want to increase the cross-sell. You find out that they want to place advertisements for additional products whenever something is added to the basket. This information is vital for you, as other solutions could also have been based on Email advertisements.
- You check other Web shops and look at their solutions.
- You frame the problem to predict which advertisements should be placed based on past shopping behavior of the current customer, past shopping behavior of all customers, and the current content of the shopping basket.
- You identify two relevant stakeholders. 1) The owner of the Web shop, who wants to increase the sales. 2) The customers of the Web shop who want to buy relevant products and have a good user experience. Irrelevant advertisements may lead to a decrease in user experience, while relevant advertisements may even improve the user experience.
- You do not identify relevant pain points in the current operation. The goal is not to solve an existing problem but only the optimization of the revenue of the Web shop.
- From the above, you identify two objectives:
- Increase the number of sales.
- Improve user experience through the placement of relevant products only.
- You will check the objectives by an evaluation of the increase in revenue through predictions and an evaluation of the customer satisfaction. The project is successful if the revenue increases by at least 5% and the customer satisfaction does not decrease. A drop of the customer satisfaction which reduces the revenue is the main risk of the project.
- The available data are mainly customer transactions, i.e., which products where bought together by customers including the data of shopping. The data is stored in a relational database. Other data is not available.
- You formulate three hypotheses. 1) Products which were frequently bought together in the past, will be frequently bought together in the future. 2) There are seasonal patterns in the sales (e.g., summer clothing, winter clothing), which are relevant for the recommendations. 3) The category to which items belong is relevant for the cross-sale, especially the brand and the type of clothing.
- You find that the resources that are available are sufficient for a pilot study that evaluates the feasibility of such predictions for cross-sell. However, an assessment of how this will affect the user experience as well as a roll-out into production cannot be achieved with the resources available and would have to be done in a separate project.

Data Preparation

After the discovery the technical work on the project starts with the data preparation. The data preparation has two major goals: 1) the preparation of the infrastructure for the data analysis and the loading of all relevant data into that infrastructure; and 2) gaining an in-depth understanding of the data.

The effort for the preparation of the infrastructure is somewhere between writing a few lines of code and a huge effort that consumes several person years of resources. If the data is relatively small and easy to access, e.g., through a single SQL query or by loading the data from a comma separated value file, this is trivial. However, if you are dealing with big data, if you also have to collect the data, or if the access to the data is difficult due to some other reason (e.g., data privacy concerns), this can be quite difficult and may require lots of effort.

The general process for getting the data into the infrastructure is called *ETL*: extract, transform, load. First, the data is extracted from where ever it is currently stored. This means writing code for loading data from files, databases,

or potentially collecting the data from other sources through tools, e.g., through Web scraping. Once the data is extracted it is transformed into the required format. This transformation usually includes quality checks, e.g., to filter data with missing values or data that contains implausible values that are likely wrong. Moreover, data must often be (re-)structured and converted into different formats. For example, content from blog posts may have to be split into different fields, e.g., title, content, and comments, character encodings may have to be harmonized, and time stamps might need to be converted into a common format. Once all this is done, the data can be loaded into the analysis environment.

A variant of ETL is to switch the transformation and the loading, i.e., *ELT*. In this case, the raw data that is extracted and loaded directly into the analysis environment and all subsequent transformation are already performed inside the analysis environment. Whether ETL or ELT is a better choice depends on the use case. A common argument for using ELT instead of ETL is that the transformations may be so complex, that they require the computational power of the analysis environment. Moreover, ELT allows the evaluation how different transformations influence the results, because they can be changed flexibly, including access of the raw data for the analysis. A common argument for ETL over ELT is that transformations may be too time consuming to perform them possibly repeatedly after reloading the data.

The second major aspect of the data preparation is to get an in-depth understanding of the data. For this, the data scientists must study the available documentation about the data and apply the domain knowledge to understand what the data means. Ideally, the data scientists know the meaning for every single type of data there is, e.g., every column in a relational database, or what different document types there are in a text mining problem and how a structure can be imposed on this unstructured data. This type of work can be categorized as understanding the *meta data*, i.e., the data about the data.

However, the data should also be considered directly, i.e., the data should be *explored* - an activity that is tightly coupled with the transformations of ETL. This means that data scientists should consider relevant statistical markers and visualize data (Chapter 3). The goal is, e.g., to understand the distribution of numeric data, identify invalid data, determine and remove differences in scales of the data for further harmonization. Additionally, data scientists should try to identify which data they actually need and which data may be removed. While dropping irrelevant data early carries the risk that data is dropped that may actually be useful, it can also be of great help if the volume of the data is reduced significantly. Data scientists should always assess this trade-off.

At the end of the data preparation, all relevant data should be available in the analysis environment and relevant pre-processing steps to transform the data into the required representation should have been performed.

Example (continued)

The sales data is stored in a relational database and consists of 352,152 transactions. Each transaction has on average 2.3 items that were bought and is associated with a time stamp in the ISO 8601 format, as well as the anonymized identifier of the user who bought the items. A separate table stores additional information about the items, e.g., the price, as well as a list of categories to which the item belongs (e.g., male clothing, female clothing, trousers, sweater, socks, brand). There is also additional data available, e.g., the payment type, which you decide to drop for your analysis because you do not expect a reasonable relationship to cross-sell.

The overall volume of the data is about one Gigabyte. You decide to use an ELT process, because loading the from the database only requires about one minute and you can then flexibly explore the data while you define the required transformations.

During the data exploration you identify 2,132 transactions without items, which you drop because these are invalid data. Moreover, you note that certain brands are bought very infrequently. You decide to merge all these brands into a new category “Other brand”.

You decide to create four different representations of the transactions to facilitate using different information in the downstream analysis:

- The items as is.
- The items replaced with the type of the clothing (socks, . . .).

- The items replaced with the brand.
- The items replaced with the combination of type and brand.

Model Planing

The goal of the model planning is to design the models for the data analysis. This means that you must decide how exactly you want to analyze the data and which models you want to use to gain insights. The models must be suited both for the use case and the available data. There are different aspects that influence the candidates for suitable models. The goal of the analysis usually prescribes the type of model that may be used.

- Association rule mining can be used if the goal is to identify rules that describe relevant relationships within transactions (Chapter 5).
- Clustering can be used to identify groups in a large set of data (Chapter 6).
- Classification can be used to predict to which category objects belong (Chapter 7).
- Regression can be used to describe relationships between variables and to predict numeric values (Chapter 8).
- Time series analysis can be used to analyze temporal data (Chapter 9).

There are different ways to build, e.g., classification models. There are additional considerations which can help with the choice of a suitable model. For example, it may be important to gain a detailed understanding of why a model predicts a certain outcome. In this case, a *white box* model must be used, where the inner workings are understandable for the data scientists - and possibly also domain experts. In case understanding of the inner workings of a model is not necessarily required, *black box* models can also be used. For most problems, black box models (e.g., neural networks) have a higher performance ceiling than white box models (e.g., decision trees).

Another consideration for the choice of model is the volume of the data and the computational resources that are available. Not every model scales equally well with large amounts of data. Moreover, some models work only with massive amounts of data, others work better with smaller amounts of data. Because all this must be considered, data scientists should have a broad knowledge about different modeling approaches, because there is no silver bullet that can solve all problems.

However, the selection of the model that should be trained is only a small part of the model planning. Depending on the data, you may also have to model the features that your models use and select which features you want to use (more on features in Chapter 4). You should also plan how you want to evaluate the results, e.g., how you want to score different models, if and how you want to statistically compare different models you evaluate, and how you may use visualization to support the analysis of the results. Finally, you have to decide how you want to use your data. For many use cases, data needs to be separated into different sets, i.e., training data, test data, and possibly also validation data (more on this also in Chapter 4). For big data, it is usually also a good idea to extract a small sample of "toy data" that you can use locally to test your programs

Example (continued)

You decide to use an association rule mining approach (apriori algorithm) to identify items that are frequently bought together and you want to evaluate how well the approach works on the different data sets you prepared. For the cases where you replaced the items with categories, you want to create a regression model (random forest regression) that predicts scores to model which concrete items are often bought, given that products from other categories are already in the basket. Depending on the results, you already consider that you may need to improve your models in a second iteration, as you see a risk that only cheap products will be recommended, which may lead to cross-sell, but without a strong increase in revenue. You will create visualizations that show which products are often bought together in form of a graph. You split your data based on the time stamp and use the oldest 50% as training data, the next 25% as validation data, and the newest 25% as test data.

Model Building

The model building means that you implement the code to create the models you planned from the data you have. Often, there are multiple iterations, i.e., after the models are built and initial evaluation is done. Based on these preliminary results, there might be another cycle of model planing and model building, to further improve the models and mitigate problems that may have been detected. However, when this is done iteratively, it is important that the test data is not used multiple times. The test data may only be used once, i.e., after the last time the models were built. Otherwise, the test data degenerates into validation data (Chapter 4).

The main reason the model building is a separate phase from the model planning is that there are use cases where this can be very expensive and time consuming, e.g., in case a huge neural network is trained with distributed resources in a cloud system.

> **Example (continued)**
>
> You decide to implement the model in Python using pandas, scikit-learn, and mlextend. The model can be trained and evaluated on a normal workstation.

Communication of Results

At some point, a decision must be made that the cycle of model planning and model building stops and the final results based on the test data are generated. Once these results are available, they must be presented to the relevant stakeholders. The main question that must be answered is whether the objectives are achieved or not and, consequently, if the project was successful. Moreover, key points about the results should be made, e.g., not only if an approach may be cost efficient, but also an estimate of the expected Return on Investment (ROI). Other interesting key points may be unexpected findings, e.g., relationships within the data that were not expected by the domain experts. For research projects, make clear how your approach advances the state of the art, beyond just being a couple percent better in some performance metric.

> **Example (continued)**
>
> You found that a recommendation system based on association rules for categories in combination with the prediction of the most suitable items through regression performs best. You estimate that the sales may increase by 10% generating 6% more revenue. Moreover, you found that the models only work reliably, if you separate the items by the gender they are targeting in the regression model. Otherwise, the model would nearly always predict female clothing.

Operationalization

If the project delivered good results, it may be decided to put the result into an operational product. This usually means a significant effort must be spent in the re-implementation of the model, such that it runs in a scalable way in the production environment. This might mean that the whole code must be rewritten, e.g., to be running directly within a database.

When a model is migrated into operation, this should be done with care. Ideally, it would be possible to start with a smaller pilot study. The pilot serves as additional safety net and should be used to evaluate if the expectations derived from the results on the test data really generalize to the application of the approach in the real world. Differences may surface, e.g., because users do not act on predictions as expected or simply because the behavior changed between the collection of the test data and the current state.

This is also a vital issue that should not be neglected when approaches are operationalized: because data ages, models age as wells. This means that they may become outdated and the performance may deteriorate over time. To counter this, a strategy for the re-training of models should be defined, that includes when this is done, how and by whom this is done, how the newer models will replace the current models, and how it will be measured if the performance regresses.

Example (continued)

Your customer was satisfied with the results of the project and decided to operationalize the approach and run a pilot study. For the pilot study, the predictions are shown to a randomized group of users. The pilot shall run for one month. Regardless of the survey, each user is invited to participate in a user survey regarding the user experience of using the Web shop. If the results of the user survey indicate no drop of the user satisfaction and there was a sufficient amount of cross-sell, the approach will be adopted for all customers.

2.2 Roles within Data Science Projects

Within every process, people fulfill roles to execute a project. Meriam-Webster defines roles as follows.

Definition of Role

A function or part performed especially in a particular operation or process.

Thus, roles define the responsibilities of the people working within a process. In practice, roles are often related to job titles, e.g., "Software Engineer", "Project Manager", or "Data Scientist". In general, there is no one-to-one mapping between roles and people. One person may fulfill several roles and one role may be fulfilled by several people. For example, there may be multiple people with the role "Data Scientist" in a project, and people with the role "Data Scientist" may also have other roles, e.g., "Project Manager" or "Database Administrator".

We discuss seven roles within data science projects. Please note that this is likely not sufficient, especially for larger projects, or for the operationalization of projects. In this case, additional roles may be required, e.g., software engineers, software architects, cloud architects, community managers, and testers. Moreover, we discuss the roles from an industrial perspective. However, each role has academic counterparts, which we also mention. The following tables lists the roles and their descriptions.

2.2.1 Business User

The business users are those that will be directly using the potential results of your project. Business users are important stakeholders and domain experts that should be consulted to better understand the data, the value of results, as well as how results may affect business process if they are operationalized. In academic projects, business users are anyone who may directly use your research in their daily work, either directly, or indirectly.

The business users are usually not part of the day-to-day business within projects.

2.2.2 Project Sponsor

Without the project sponsor, there would not be a project. They are responsible for the genesis of the project and provide the required resources. This may be upper management or an external customer. The sponsors are also part of the evaluation of the project success. They determine if the project was successful or not based on the final results. They also determine if additional funding should be made available and usually also decide if a project is operationalized. In academia, the projects sponsors are the principle investigators of projects, usually professors and post-docs.

The project sponsors are usually not part of the day-to-day business within projects.

2.2.3 Project Manager

The project manager is the organizer of the project work and the day-to-day business. This includes the planning and management of resources (monetary, human, computational), as well as the controlling of the project progress. The project manager ensures the milestones are achieved and objectives are met on time and with the expected quality. The project manager regularly assesses the risks and possibly implements corrective measures to mitigate the risks. In the extreme case this may also mean that the project manager has to abort projects or request additional resources, because a successful completion of the project may not be possible with the required resources. In academic projects, the project manager is often the same person as the project sponsor, or someone working for the project sponsor, e.g., a post-doc or a PhD student working in the lab of a professor.

2.2.4 Business Intelligence Analyst

The business intelligence analyst is the traditional business intelligence role. The business intelligence analysts are usually also domain experts, although not business users. Additionally, they know the data very well and are experts on creating reports from the data, including the key performance indicators (KPIs) relevant for the domain. This role does not really exist in academia. The closest thing would be a researcher, who is, e.g., used to evaluating taxonomies manually, but not through algorithms.

2.2.5 Data Engineer

The data engineer is involved in the ETL/ELT of the process and responsible with making the data accessible in the analysis environment in a scalable manner. Data engineers require strong technical skills if data still needs to be collected and in big data projects. In most academic projects, there is no separation between the data engineer and the data scientist. In some disciplines, e.g., genomics, data engineers may exist in the role of preparing genomic data from the lab for use with bioinformatics tools or in high energy physics, where whole teams prepare the huge amounts of data collected for further analysis.

2.2.6 Database Administrator

The database administrator supports the data engineer and the data scientists through the deployment and administration of the analysis environment for the project. The tasks include the installation and configuration of (distributed) databases, software for distributed computing in compute clusters, and the provision of the required tools for the analysis. In academic projects, this role is not separated from the data scientist, or the administrator of the research institute or related compute center fulfills this role.

2.2.7 Data Scientist

The data scientist is the expert with respect to the analysis of the data and modeling. The data scientist has deep knowledge about data analysis techniques and is able to select suitable techniques to achieve the desired business objective. The data scientist is responsible for the analysis of the data and has to ensure that the analysis objectives are met. If this is not possible, the data scientists must communicate this as soon as possible to the project manager, as this is usually a major risk within data science projects. In academia, this role is usually the researcher who executes a project.

2.3 Core Deliverables

A deliverable is a tangible or intangible good or service produced as a result of a project. The required deliverables are often specified as part of contracts and often associated with project milestones. Deliverables must meet the stakeholder's expectations for the project to be successful. For a data science project, there are four core deliverables that are usually required.

2.3.1 Sponsor Presentation

The sponsor presentation is for the big picture and a non-technical audience, e.g., business users, project sponsors, and the project management. The focus of the sponsor presentation should be on clear takeaway messages that are related to the business use case. For example, the performance of a model should not be given in some machine learning metrics, but rather in business metrics. This means that you do not mention "true positives", but "the percentage of correctly identified defaulting customers". The presentation should be designed to support decision making. What exactly this means depends on the state of the project and the audience. For business users, the presentation may be tailored such that improvements of the business process are highlighted to motivate the business users to adopt the approach. For project sponsors, the goal may be to aid their decision whether the results of the project should be operationalized or not.

Visualization should be simple and on point. Bar charts, line charts, and simple histograms. In case of histograms, the bins should make sense for the business users. The visualizations should support the results regarding the key performance indicators. Detailed visualizations that present in depth how and why your models work should not be used within a sponsor presentation.

2.3.2 Analyst Presentation

The analyst presentation is aimed at a technical audience, e.g., fellow data scientists. This presentation should also cover the big picture, especially key performance indicators should also be presented. However, the analyst presentation should also cover how the data was analyzed. This should include the general approach that was used, but also concrete algorithms. Depending on the audience, and the available time, it may even be a good idea to include relevant implementation details. In general, the presentation should not focus on "the obvious". If standard techniques are used, there should not be lengthy description of them. Instead, the presentation should focus on interesting aspects, e.g., unexpected problems, creative solutions, new inventions to improve performance and similar aspects.

Moreover, the presentation should cover how the project results go beyond the state of the art, especially in research presentations. However, you may also address in industrial use cases how your project would change the current state of practice and how business processes need to be adopted and may possibly be improved.

The analyst presentation can be more complex. This also means that you can use more complex visualizations, for example box plots, density plots, ROC curves, and error bars for confidence intervals. Regardless, this is not a free pass for arbitrary complex graphics. The presentation should still be clean and not overloaded.

2.3.3 Code

The developed model in form of code is usually also a deliverable. The developed code is often prototypical, e.g., not a cleanly coded library that can easily be included and re-used, but rather a couple of files with scripts that can be executed. In the worst case, these scripts may even contain workstation depend code, like absolute file paths. Clean code is often a secondary concern, the focus on the code is usually on creating a good model and the estimates of the performance.

The code is at the same time the exact specification of the models. Therefore, the code - regardless how "hacky" - is a valuable resource. The code may be re-used, re-written, or adopted or cleaned up. This can be especially valuable

during the operationalization. When the code is re-written, the old code can also be used for comparison, e.g., to check if the results are unchanged.

2.3.4 Technical Specifications

The code does not live on its own. Code requires documentation and specifications. This documentation should describe the environment in which the code can be executed, e.g., which operation systems are supported, which other software is required, how dependencies can be installed, and how the code can be run. This is, unfortunately, often neglected. Technical specifications are often missing, outdated, or incomplete. This lack of documentation may mean that the code becomes unusable. Regardless, good data scientists ensure that this is not the case. In research, this may mean that a replication package is provided together with a publication that contains all code and additional documentation required for running the code. In industrial projects, this should be a standard part of the packaging of the source code.

2.3.5 Data as Deliverable

When data is collected or significantly pre-processed and cleaned up during a project, the data itself may also be deliverable. Sharing of the data may enable future projects and help with the testing of re-implementations during the operationalization. Data sharing is especially important in academia. Without the sharing of data, research cannot build on each other. Moreover, replications and the reproduction of prior results may not be possible.

How data can be shared depends on the sensitivity and the amount of data. Sharing of big data is difficult, as it may actually be impossible to create download links. In this case, the data may be shared through multiple smaller packages or by providing an access point to the database that allows external querying of the data. If the data contains personal data, sharing may not be simple and require dedicated data privacy agreements or prior anonymization. In the industrial context, data can be shared internally within a company. In research, data sharing should follow the FAIR. Thus, the data must be findable (e.g., with a Digital Object Identifier (DOI)), accessible (authentication support if required), interoperable (provides all information required for use by others), and reusable (has a license and is in line with community standards). Ideally, not only the data is shared, but also the protocols and tools used to create the data.

CHAPTER

THREE

DATA EXPLORATION

3.1 Overview of Data Exploration

The goal of the data exploration is to *learn* about the data. This means that the data scientist must understand the structure of the available data as well as the values of the data. At the end, the data scientist knows the basic characteristics of the data, e.g., the structure, the size, the completeness (or rather where data is missing), and the relationships between different parts of the data.

There are many methods for data exploration. The exploration is usually a semi-automated interactive process in which data scientists use many different tools to consider different aspects of the data. The first element of the toolbox of a data scientist are simple text editors and command line tools like head, more, and less. These tools allow the data scientist to inspect non-binary raw data, e.g., comma-separated values (CSV):

```
!head 03_Data-Exploration.ipynb
```

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# Data Exploration"
   ]
  },
  {
```

From this, we learn that the ipynb format used by Jupyter notebooks uses the JavaScript Object Notation (JSON) to store data as a list of cells, that have a type, metadata, and associated source code. Without any additional tools, just through the simple head command that list the first lines of a text file.

> **Note:**
>
> You can run any shell command (e.g., bash) if you prefix the name with an exclamation mark directly from a Jupyter Notebook. Thus, `!head` in a Jupyter Notebook is the same as typing `head` in the command line. Whether this works or not depends on the command line. Since the standard command line of Windows does not have a `head` command, this may not work.

Of course, this is not the only way to learn about how the data is stored by Jupyter Notebooks. It is also not clear if all information (e.g., allowed fields) are visible, because the current file may not use all possible features of the ipynb format. Therefore, it is also important to look at the metadata, in this case the description of the format. Good documentation of metadata is usually much more detailed and contains all aspects of the format.

The metadata is not only restricted to file formats, but also other aspects, e.g., where data in a column of a relational database comes from (i.e., the source of the data), where related information may be found, and other useful information about the data. What the metadata does not contain (except maybe example values) is information about the

values in your dataset. However, learning about these values is also vital to understand how a good model for your use case, based on the data, may look like. In the following, we look at two commonly used methods for learning about the values of data, i.e., summary statistics and visualizations.

3.2 Descriptive Statistics

Descriptive statistics try to summarize data in single values that describe properties of the data. This is not to be mixed up with *inductive* statistics, that allows the prediction of properties of the data. Consequently, you cannot use descriptive statistics to create reliable forecasts of the data.

Within this chapter, we cover the following descriptive statistics:

- The *central tendency* of data, described by the *arithmetic mean*, *median*, or *mode*.
- The *variability* of the data, described by the *standard deviation* or *interquartile range*.
- The *range* of the data, described by the *min* and *max* value.

There are other descriptive statistics for the properties above, that we do not discuss in this chapter, e.g., the *trimmed mean* and the *harmonic mean* for the central tendency, or the *median absolute deviation of the median* for the variability. Moreover, there are also descriptive statistics for other properties of the data, e.g., for the *shape* (kurtosis, skewness). However, these are out of scope of this chapter and you can mostly do without them, especially if you combine descriptive statistics with visualizations.

We use the notation $x = x_1, ..., x_n$ with $x_1, ..., x_n \in \mathbb{R}$ in this section. This means that x is a set of real valued numbers. If x is only a subset of all possibly data, x is called a *sample*.

3.2.1 Central Tendency

The central tendency of data is a statistical marker that should be a *typical value* in the center of the data, i.e., in the middle. This does not mean that many data points will have that exact value. Instead, the central tendency marks the "middle" of the data, i.e., its central location. For comparison, consider cities. They also have a center, which is typically somewhere in the middle of the city.

There are different ways to measure the central tendency of the data. The *arithmetic mean* is defined as

$$mean(x) = \frac{1}{n}\sum_{i=1}^{n} x_i, x_i \in \mathbb{R}.$$

The arithmetic mean can easily be described in the natural language. It is the sum of all values divided by the number of values. The arithmetic mean is a good choice to describe the central tendency of data that follows a normal or a uniform distribution. In general, the arithmetic mean works reasonable well as long as 1) the data is without "holes", i.e., without large intermittent regions without values, including outliers, and 2) the data is symmetric around the arithmetic mean, i.e., the *spread* of the data on the left and the right side of the mean is similar. Thus, the arithmetic mean only works well if assumptions on the distribution of the data are met. Statistical methods that make such assumptions are called *parametric* statistics and may be misleading in case the assumptions are not fulfilled. We will show examples that demonstrate how the mean value can be misleading through *Anscombe's quartet*.

An alternative to the mean value is the *median* which is defined as

$$median(x) = \begin{cases} \bar{x}_m & \text{if } n \text{ is odd with } m = \frac{n+1}{2} \\ \frac{1}{2}(\bar{x}_m + \bar{x}_{m+1}) & \text{if } n \text{ is even with } m = \frac{n}{2} \end{cases}$$

where $\bar{x}$ are the values of x sorted by their size. Thus, the median is the middle value in the data: 50% of the data is less than or equal to the median, 50% of the median is greater than or equal to the median. The median is a *non-parametric*

statistic, i.e., the median works for any distribution of data. Thus, the median can be used instead of the arithmetic mean if the assumptions on the distribution for the arithmetic mean to be meaningful are not met.

What it means that the median is more robust than the arithmetic mean can be demonstrated with a simple example. Consider the arithmetic mean and median of the following data, that fulfills the assumption of the arithmetic mean:

```
import statistics  # we use the statistics from the Python standard library

data = [8.04, 6.95, 7.58, 8.81, 8.33, 9.96, 7.24, 4.26, 10.84, 4.82, 5.68]

print('mean:  ', statistics.mean(data))
print('median:', statistics.median(data))
```

```
mean:   7.500909090909091
median: 7.58
```

Both values are relatively similar, both represent the central tendency of the data well. Now we add a single very large outlier to the data.

```
data.append(100)
print('mean:  ', statistics.mean(data))
print('median:', statistics.median(data))
```

```
mean:   15.209166666666667
median: 7.81
```

The arithmetic mean is now a lot higher and actually larger than all but one data point. This is not a good representation of the central tendency of the data. The median did not change a lot and remains almost the same. This is because the median only skips to the next sorted value, which is robust against such outliers.

You might ask yourself why you should use the arithmetic mean at all, if the median is apparently superior because it is more robust. The biggest reason for this comes from stochastics: the arithmetic mean is closely related to the expected value of random variables. Moreover, the normal distribution can be fully described by the mean value and the standard deviation. Thus, for normally distributed data, the mean value is actually the perfect estimate for the central tendency. Regardless, if you are unsure about the distribution of the data and fear that there might by outliers, it is safer to use the median instead of the arithmetic mean.

The arithmetic mean and the median can only be used for numeric data. However, what if you have, e.g., data about cloth sizes? These are not numeric values, but rather something like “small”, “medium”, and “large”. Such data is categorical data and further discussed in Chapter 4. The *mode* can be used to calculate the central tendency of such data. The mode is defined as the value that occurs the most in a sample x.

```
data = ['small', 'medium', 'small', 'large', 'large', 'medium', 'medium']
print('mode:', statistics.mode(data))
```

```
mode: medium
```

Please note that while the mode may work well for categorical data or discrete numeric data, it can also be misleading if the data is *bimodal* or *multimodal*, i.e., data where multiple values are all occurring frequently. The extreme case would be data where the categories are uniformly distributed. Because each category is equally likely, the mode would be completely random. Moreover, the mode does not work for continuous numeric data, because here it is very unlikely that the same value occurs twice.

3.2.2 Variability

The variability measures how far the data is spread out. A small variability means that most data is close to the central tendency of the data. A high variability means that the data is spread out over a large range.

The most common measure for the variability is the *standard deviation*, which is defined as

$$sd(x) = \sqrt{\frac{\sum_{i=1}^{n}(x_i - mean(x))^2}{n-1}}.$$

The standard deviation is the square root of the arithmetic mean of the difference from the arithmetic mean, except that the division is by $n-1$ instead of n. This is due to something called *degrees of freedom*, which we do not discuss here further. The standard deviation tells us how much deviation from the mean value we can expect. The standard deviation works well as a measure for the variability, when the mean value is a suitable representation for the central tendency. This means that the standard deviation is also a *parametric* measure for the variability, that can also be misleading.

Same as the median is the non-parametric counterpart to the arithmetic mean, there are also non-parametric statistics for the variability. The *interquartile range* is defined as

$$IQR(X) = Q_{upper} - Q_{lower}$$

where Q_{upper} is the upper quartile and Q_{lower} is the lower quartile. The upper quartile and lower quartile are defined analogously to the median, but with 75% and 25% of the data. This means that for the upper quartile, 75% of the data are less than or equal to Q_{upper} and 25% are greater than or equal to Q_{upper}. For the lower quartile, 25% of the data are less than or equal to Q_{lower} and 75% of the the data are greater than or equal to Q_{lower}. It follows that at least 50% of the data is within the interquartile range.

In general, you should not use the standard deviation, when you use the median. Below is the example we already used to demonstrate that the arithmetic mean is not robust with respect to outliers.

```
from scipy import stats  # we use scipy for the IQR

data = [8.04, 6.95, 7.58, 8.81, 8.33, 9.96, 7.24, 4.26, 10.84, 4.82, 5.68]
print('Without outlier')
print('sd: ', statistics.stdev(data))
print('IQR:', stats.iqr(data))

data.append(100)
print('With outlier')
print('sd: ', statistics.stdev(data))
print('IQR:', stats.iqr(data))
```

```
Without outlier
sd:  2.031568135925815
IQR: 2.255000000000001
With outlier
sd:  26.77235292350312
IQR: 2.465
```

Regardless, if the assumptions for the standard deviation are fulfilled, this is a very good measure for the variability of data. Same as for the mean value, this is especially true for normally distributed data. Together with the mean value, the standard deviation describes the normal distribution. For the normal distribution it can be shown that 68% is no more than one standard deviation away from the arithmetic mean, 95% of the data is within two standard deviations of the mean, and 99.7 percent is within 3 standard deviations of the mean. This is also known as the 68-95-99.7 rule. A weaker statement is also possible for other distributions. Chebychev's inequality holds for all distributions with a finite expected value (can be thought of here as arithmetic mean) and finite non-zero variance (the variance is the square of the standard deviation). It follows from Chebychev's inequality that 50% of the data is no more than $\sqrt{2} \cdot std$ away from the expected arithmetic mean.

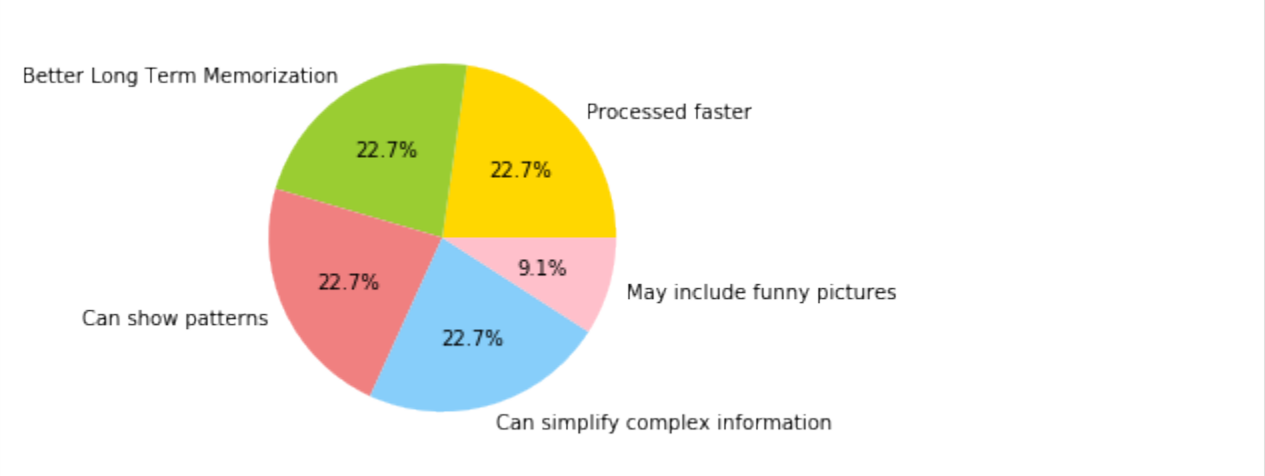
3.2.3 Range

The range of data describes which values can possibly be observed. For numeric data, this is defined by the minimum, i.e., the smallest observable value and the maximum, i.e., the largest observable value. Both can easily be estimated from data by simply defining the range as the the interval between the minimal *observed* value and the maximal *observed* value.

The range may sound trivial as statistical markers. However, the range is a surprisingly powerful tool to identify simple problems in the data, that may be hidden by other descriptive statistics we described above. Especially invalid data that is outside a reasonable range can be easily seen. For example, if you have the age in years, and observe a minimum value of -1 this is implausible and indicates that this may be a missing value. If you have a maximum of 130 years, this is also not possible because nobody lived that long (so far). Since both problems would be relatively close to real values, i.e. not outliers, this would likely not be detected by analyzing the central tendency and the variability.

3.3 Visualization for Data Exploration

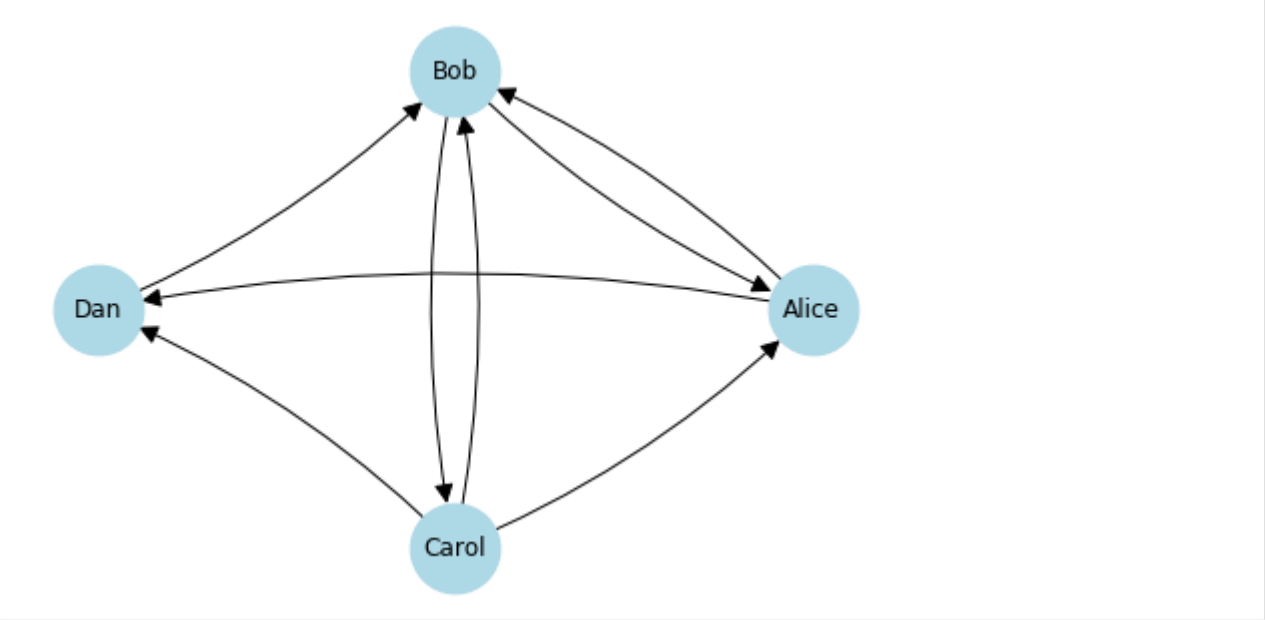
Visualization is a powerful tool to learn about and understand data.



There are several advantages of visualization in comparison to other forms of showing data, e.g., tables and descriptive statistics. In general, we are faster at processing visual information. Thus, we can understand more about data in the same amount of time. Moreover, most people are better at memorizing visual information than, e.g., text. From personal anecdotal evidence that was certainly not obtained using any scientific method, I believe this is true for students. In exams, I have seen many solutions where the shape of a diagram was correctly drawn from memory, but the annotations, i.e., the associated textual data was wrong or missing.

However, the advantages of visualizations go beyond processing speed and memorization. Visualization may also lead to knowledge about the data that would otherwise be very hard to obtain, e.g., about patterns within data of complex relationships.

Consider the following example. Alice knows Bob and Dan, Dan knows Bob, Bob knows Carol and Alice, Carol knows Alice, Bob, and Dan. The prior sentence is quite complex, hard to follow, and you do not get an intuitive understanding of the relationship between Alice and Bob. Now the sentence as a visualization using a directed graph:



The graph is easy to read and gives us an intuitive understanding of the relationships between the four actors. Please note that for this - and all other graphs in this chapter - we leave out legends, detailed styling, etc. Instead, we create the graphs with minimal code such that they allow us to get the insights we want. This is because we consider visualization as a tool for data exploration. If we would consider visualization in general, especially how you should create graphs that would be used in written text (books, scientific publications, student theses), other aspects would also be relevant, e.g., consistent font types and sizes, consistent coloring, legends, and titles.

3.3.1 Anscombe's Quartet

Anscombe's quartet is a famous example for how descriptive statistics may be misleading. The example is based on four different sets of data $(x_1, y_1), ..., (x_4, y_4)$ where with 11 values and for two variables x_i and y_i, $i = 1, ..., 4$.

$x_{1,2,3}$	y_1	y_2	y_3	x_4	y_4
10	8.04	9.14	7.46	8	6.58
8	6.95	8.14	6.77	8	5.76
13	7.58	8.74	12.74	8	7.71
9	8.81	8.77	7.11	8	8.84
11	8.33	9.26	7.81	8	8.47
14	9.96	8.10	8.84	8	7.04
6	7.24	6.13	6.08	8	5.25
4	4.26	3.10	5.39	19	12.50
12	10.84	9.13	8.15	8	5.56
7	4.82	7.26	6.42	8	7.91
5	5.68	4.74	5.73	8	6.89

When we look at the data with descriptive statistics, they look very similar.

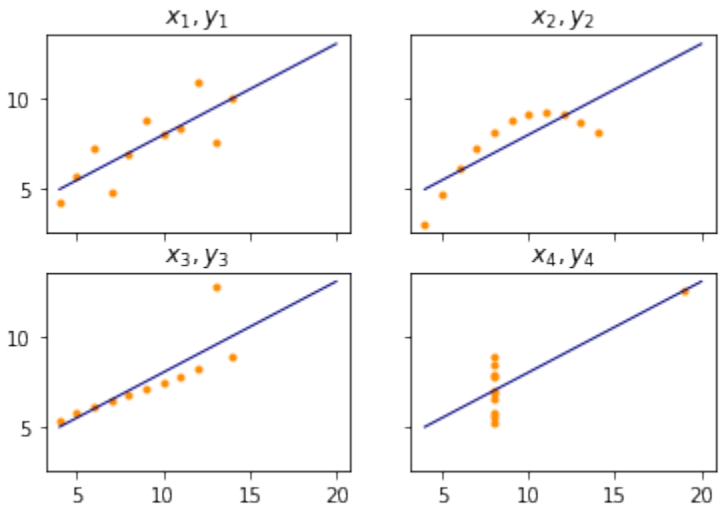
```
arithmetic mean of x (same for x1, x2, x3), x4
9
9
```

(continues on next page)

(continued from previous page)

```
standard deviation of x (same for x1, x2, x3), x4
3.3166247903554
3.3166247903554
arithmetic mean of y1, ..., y4
7.500909090909091
7.500909090909091
7.5
7.500909090909091
standard deviation of y1, ..., y4
2.031568135925815
2.0316567355016177
2.030423601123667
2.0305785113876023
```

The data even shares more similarities. If we were to fit a linear regression (Chapter 8) of y_i using x_i, we would always find the same regression line, i.e. $y = 3 + 0.5 \cdot x$. Thus, from a statistical point of view the four data sets may be interpreted as similar. However, when we visualize the data with a simple pair-plot, we see that this is not the case.



The orange points show the location of the data, the blue line shows the regression line for $y = 3 + 0.5 \cdot x$ which is optimal for the data. From the statistics, one may expect the data to look like x_1, y_1, i.e., data with a fairly linear relationship. This means that both x_1 and y_1 change relatively regularly in form of a line. The pair x_2, y_2 shows something that looks a lot like an inverse parabola. The pair x_3, y_3 is actually perfectly linear, but with one outlier which skews the regression line. The pair x_4, y_4 is even worse. x_4 is actually constant with one exception.

The example demonstrates how descriptive statistics about data can be misleading and should be used with care. However, the more statistical markers are used, the less likely it is that the results are misleading. The critical reader may also complain that at least y_3 and x_4 have outliers and that the arithmetic mean and the standard deviations are not good criteria anyways. However, the problem is generic and there are many more examples, also for more statistical markers, as the work by Matejka and Fitzmaurice demonstrates.

3.3.2 Single Features

The basic analysis of data is that of the distribution of single features within your data to understand how the values of the feature are distributed, similar to the analysis of single features with descriptive statistics. We look at single features with histograms, density plots, rugs, and box plots.

In the following, we will do this for features from the Boston housing data, a dataset about house prices in Boston that was initially published in 1978. The description of the data is the following.

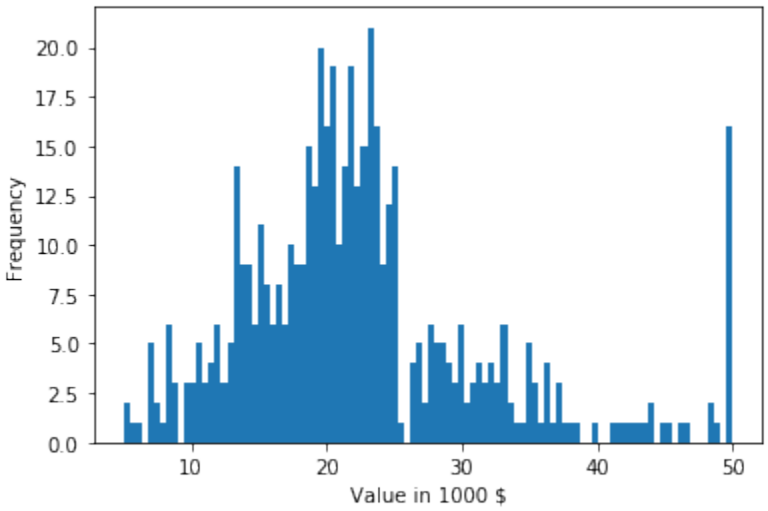
```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive.
Median Value (attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots
over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per
town
        - CHAS     Charles River dummy variable (= 1 if tract
bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10
million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior
to 1940
        - DIS      weighted distances to five Boston employment
centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion
of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in
$1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-
databases/housing/


This dataset was taken from the StatLib library which is
maintained at Carnegie Mellon University.
```

(continues on next page)

(continued from previous page)

```
The Boston house-price data of Harrison, D. and Rubinfeld, D.L.
'Hedonic
prices and the demand for clean air', J. Environ. Economics &
Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression
diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the
table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine
learning papers that address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying
Influential Data and Sources of Collinearity', Wiley, 1980.
244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based
Learning. In Proceedings on the Tenth International Conference of
Machine Learning, 236-243, University of Massachusetts, Amherst.
Morgan Kaufmann.
```

The above description is an example of metadata, in this case the documentation of the data we are using. We now take a closer look at the feature MEDV, the median value of owner-occupied homes in $1000's. *Histograms* are a simple and effective way to learn about the distribution of data.



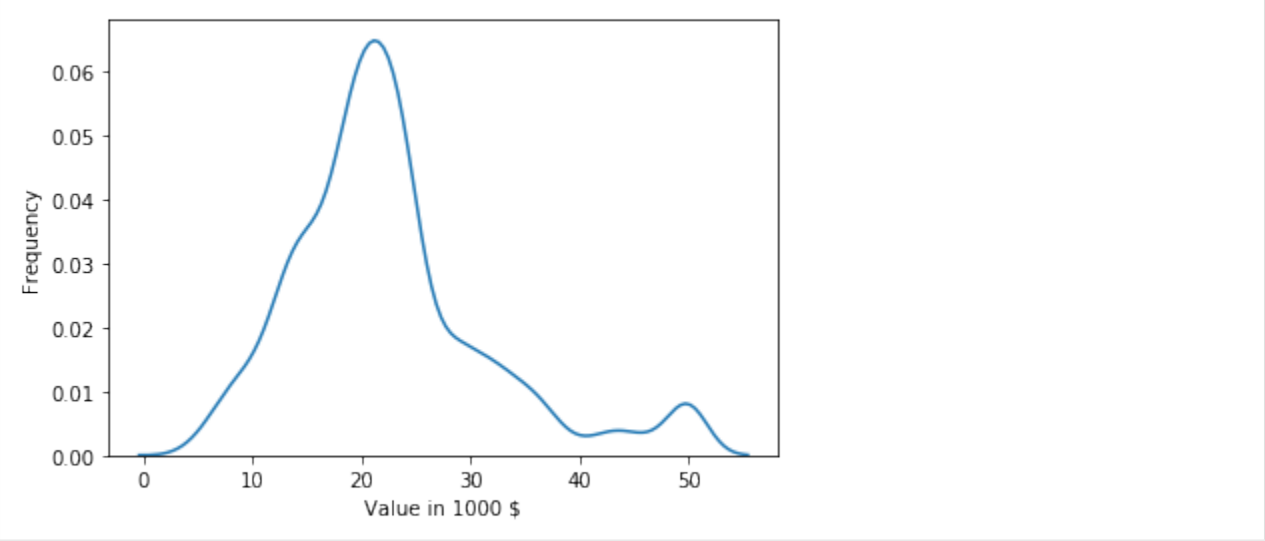
The histogram visualizes how much data is in each region of the data. For this, *bins* are created and the histogram shows how many instance are in each bin. The above example shows how many instance are in each of the 100 bins. The histogram tells us a lot about the data.

- For values below 30 the data seems to follow a normal distribution. This is indicated by the bell-shape of the histogram. The mean value of this normal distribution is somewhere around 22. This can be estimated by taking a look at where the peak of the histogram is located, i.e., where the most values are. The standard deviation is likely somewhere between 7 and 11. Such a rough estimate of the standard deviation can be obtained using the

68-95-99.7 rule we discussed *above*.

- A lot of values are in the largest bin on the right side of the plot, i.e., exactly at 50. This indicates that all values larger than 50 were set to exactly 50. Thus, while the numeric value is 50, the actual meaning is greater or equal to 50.
- There seems to be a long tail of data on the right side of the plot, i.e., there are expensive homes of all prices, but not many. Such a long tail on the right side of a plot is also called *right skew* or *positive skew*.

Another way to look at a single feature is to use a *density plot*.

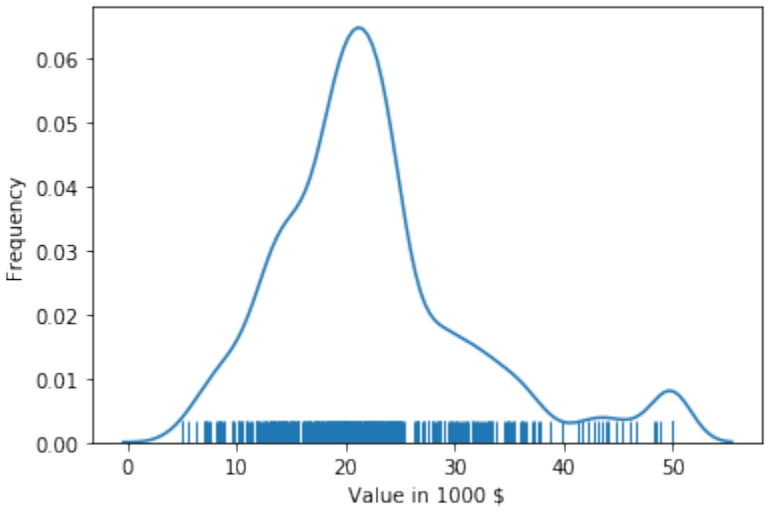


The density plot shows an estimate of the probability density function of the distribution of the feature. While it is a bit of a simplification, you can think of the density plot as a continuous histogram. An advantage of the density plot over the histogram is that it is often easier to see the distribution of the data. In the example above, the bell-shape is clearer than in the histogram. Thus, it is easier to see that the data is mostly normal. However, density plots also have two disadvantages in comparison to histograms. The smaller disadvantage is that y-axis is less meaningful. In the histogram, you can directly see the counts, i.e. how many instances in your data fall into a bin. The density plot shows a frequency that is somewhere between 0.0 and 1.0. This is more or less the fraction of the data, that is located at any specific point at the x-axis.

The second disadvantage of density plots is bigger. The histogram clearly shows the strange behavior of the data at the value 50. In the density plot, there is only a small peak that looks like there may be another normal distribution with less data and a mean value of 50. This is misleading and hides what is actually happening in the data. This is always a risk with density plots and it cannot be avoided, due to the way density plots are created. Such misrepresentations of the data are more likely if the data is less dense and at the boundary of the range in the data. Often, the density function even goes slightly beyond the range of the data.

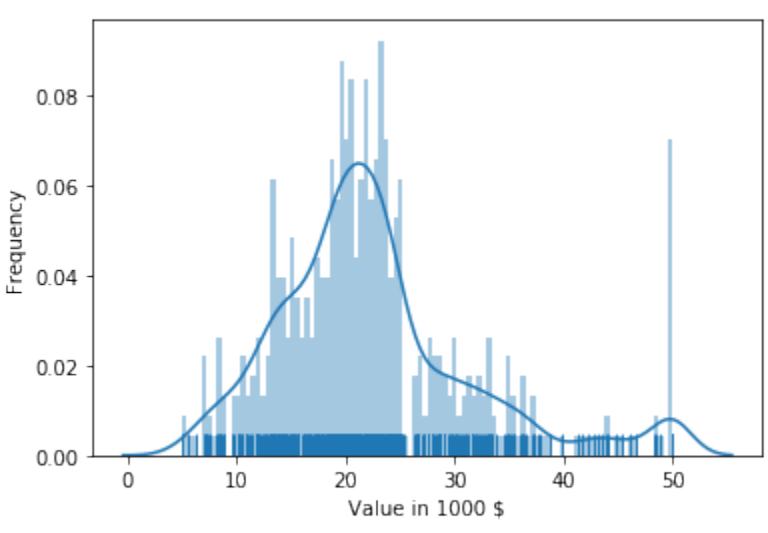
> Note:
>
> Density plots are created using a *kernel density function*, often the normal distribution. This density function is fit to each value of the data with a certain scaling parameter and bandwidth. Then, the values of all these kernel densities are summed up to create the density plot. This explains the problems at the boundaries: the kernel density function fit to the highest data point has to go beyond the range of the data, given that it is a normal distribution with a positive variance. And if there is strange behavior, e.g., due to many values being exactly the same at the boundary, this is hidden by the scaling parameter.

A simple way to avoid the problem that density plots may indicate that there is data in regions without data is to use a *rug*.



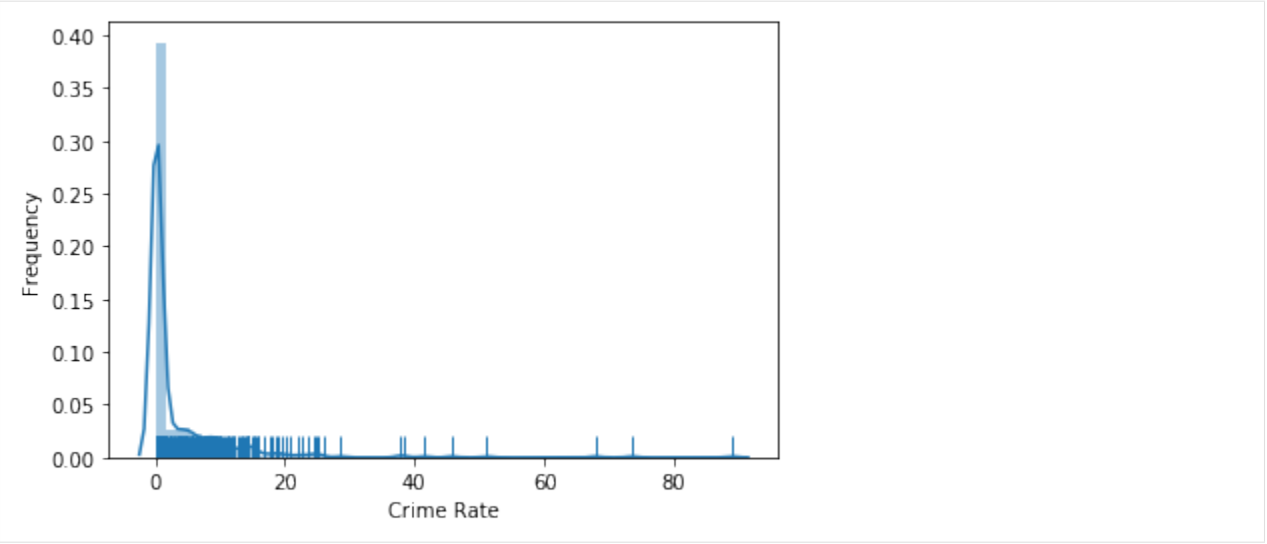
The rug shows where data points are located. In combination with the density plot, we can now see that there is no data less than 5 and greater than 50, even though the density plot indicates this. We also see that the data between 38 and 50 is relatively sparse and the rug looks almost uniform, which matches that the density plot almost looks like a line in that region - with the exception of the bump at 50. Thus, the rug gives us more information and we can avoid misinterpretations of the density plots.

Another effective way to look at a single feature is just to combine the approaches above: visualize the histogram, the density plot, and the rug together.



With that plot, we can see where the data is located and how much data we have in each bin, we get a nice visualization of the probability density function, and we see the actual location of the data in the rug. Thus, we have all advantages of both the histogram and the density plot. The only (minor) drawback is that the histogram is a bit different than the one above: it shows us the frequency as the percentage of the overall data that falls within in a bin. The reason for this is that this is the same scale on the y-axis as for the density plot. Thus, we lose the information of the counts of the values within each bin. If this is relevant, a simple histogram should be used.

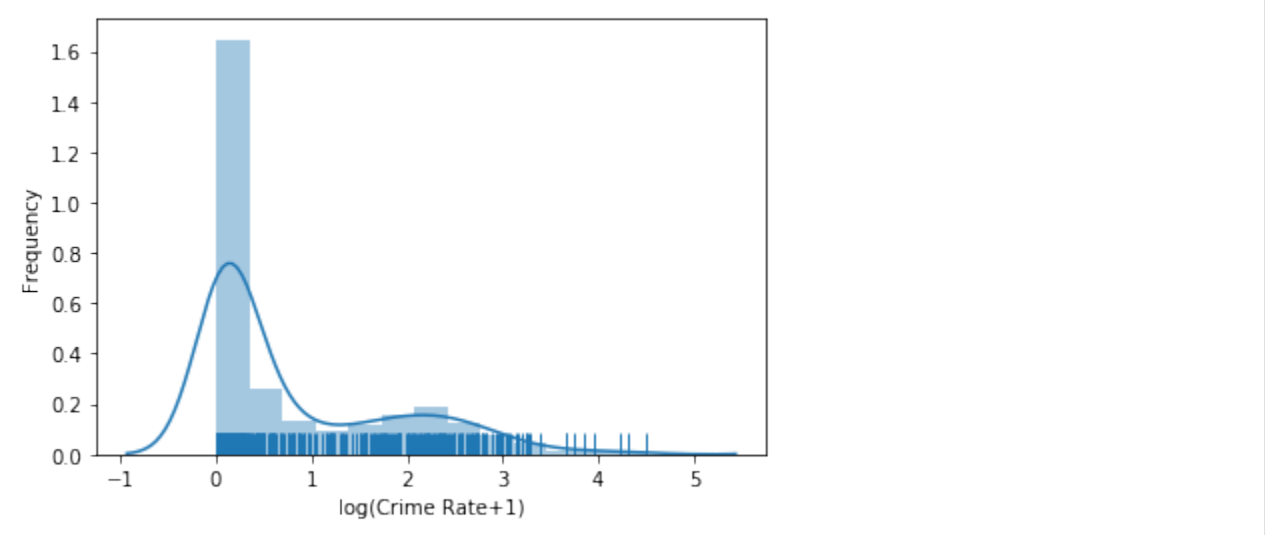
The feature MEDV that we considered above is relatively "well-behaved", i.e., mostly normal with a tail on the right side. The only problem with that feature is the grouping at 50, which can lead to problems with certain modeling approaches. Now we take a closer look at another feature from the Boston housing data: CRIM, which is defined as the per capita crime rate by town.



The plot shows that most data is very close to 0, but there is also data for almost all values up to 15, after which the data gets sparse, with some very high outliers. However, we cannot see what happens around 0, because the data is too dense. This indicates that the data *may* be exponentially distributed. Thus, there is a simple trick to solve this: use the logarithm. Instead of plotting the data directly, we plot $\log(x + 1)$. The addition of one is a smoothing factor also known as *Laplace smoothing*. Through the smoothing factor we avoid the problem that the logarithm of 0 is undefined. Moreover, because $\log(1) = 0$, it trivially follows that $\log(0 + 1) = 0$, i.e., the zero without the logarithm is actually mapped to the zero if Laplace smoothing is used.

> **Note:**
>
> Laplace smoothing is actually a bit more generic than just adding one and can also mean adding other values. The variant above is, therefore, also called *add-one smoothing*. The use of this smoothing is so common, that numpy has functions log1p for applying the natural logarithm with Laplace smoothing and exp1m for inverting the operation.

Let us look the the logarithm of the crime rate.

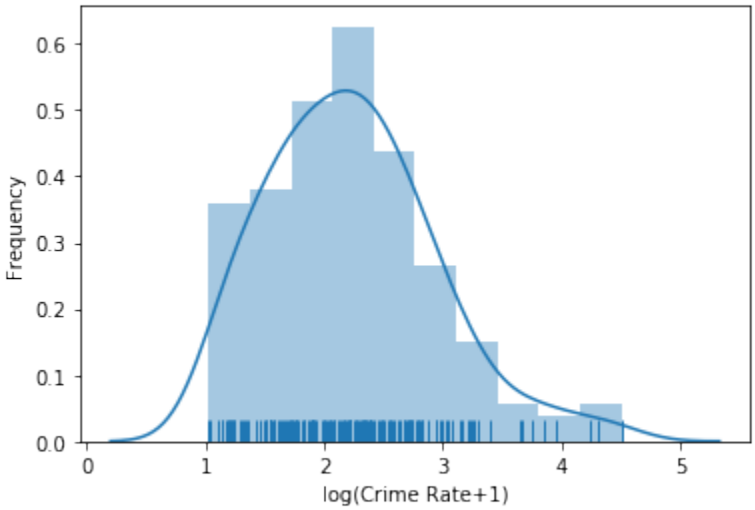


We now see more information about the crime rate. From the above plot we learn the following.

- About 80% of the areas are nearly without crime, i.e., crime rates close to zero. That most data is close to zero was already visible without the logarithm, but the ratio is now much clearer and can be seen in the frequency of

the density plot. Moreover, the drop of the crime rate from 0 to larger values up to 1 roughly looks like the right side of the bell-shape of a normal distribution.

- The remaining 20% of the areas seem to a have a *lognormal* distributed crime rate. A distribution is called lognormal, if it looks normal when the logarithm is used. This seems to be the case here. The plot looks normal starting for values greater than 1 with a mean value of around 2.2 and a relatively large standard deviation, i.e., the distribution is spread out. Please note that this is the mean value *after* the logarithm is applied, i.e., we need to invert the logarithm using the exponential function to see what this means for the actual values. Thus, the lognormal part starts at $e^1 - 1 \approx 1.7$ in the data without the logarithm and the central tendency is at $e^{2.2} - 1$. We can even take a closer look at this part of the data, if we just cut of the crime rates less than 1.7 and see that this analysis is correct.



We now use a different data set than Boston, because it is better suited to demonstrate the following plots. The data we now use is called the Iris data.

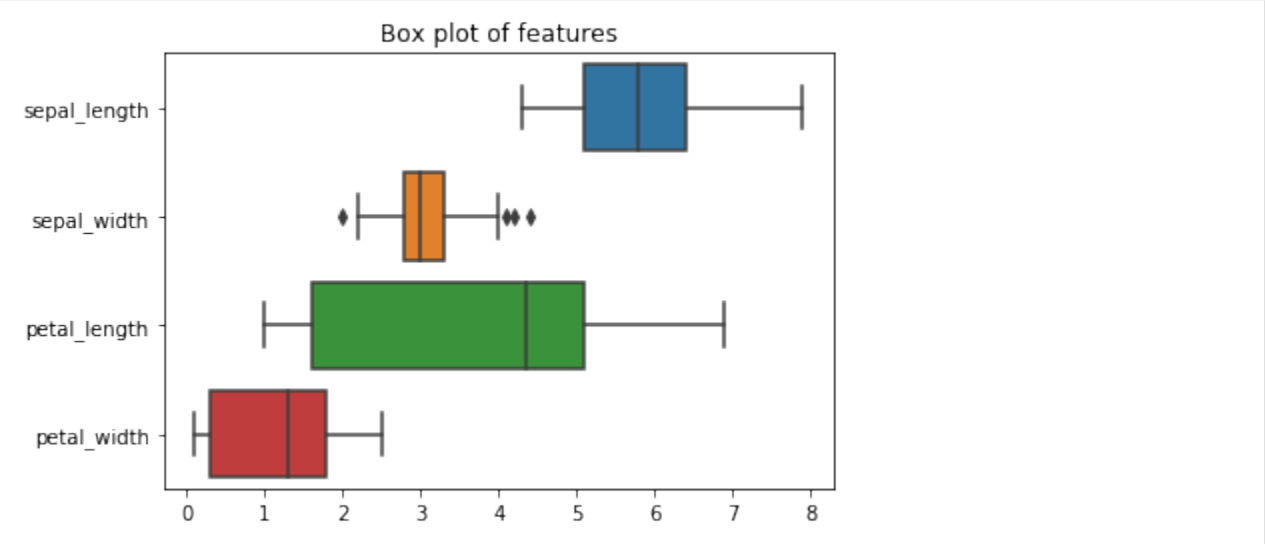
```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and
the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:
```

(continues on next page)

(continued from previous page)

```
    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The
dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as
in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in
the field and
is referenced frequently to this day.  (See Duda & Hart, for
example.)  The
data set contains 3 classes of 50 instances each, where each
class refers to a
type of iris plant.  One class is linearly separable from the
other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic
problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in
"Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and
Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page
218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A
New System
     Structure and Classification Rule for Recognition in
Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and
Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".
IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s
AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

The Iris data has four features about the sepals and petals of the plants and the information to which kind of iris each instance in the data belongs, i.e., if it is a Setosa, Versicolour, or Virginica. We now look at the data about sepals and petals with *boxplots*.



A boxplot consists of a box, to which lines are attached, the *whiskers*. These plots are also known as Box-Whisker plots. The boundaries of the box are the lower quartile Q_{lower} and the upper quartile Q_{upper}. It follows that the length of the box is the interquartile range. Within the box, there is a line. The line marks the median of the data. The definition of the length of the whiskers is not straightforward. The end of the lower whisker is the smallest instance of the data, where the value is greater than $Q_{lower} - 1.5 \cdot IQR$. The end of the upper whisker is similarly defined as the the largest value that is less than $Q_{upper} + 1.5 \cdot IQR$. The idea behind the whiskers is that every value that is more than 1.5 times the IQR away from the quartiles is considered an outlier. Thus, the whiskers show the range of the data without outliers. If there are outliers, each outlier is depicted as a point, as can be seen in the boxplot of the sepal width.

From this definition of the boxplot, we now know what we see: a visualization of the non-parametric descriptive statistics of the data. Interesting aspects of boxes are, e.g., the following.

- The locations of the boxes, i.e., in which range is most of the data located.
- The length of the boxes, i.e., how spread out is the central data.
- The length of the whiskers, i.e., the range of the overall data.
- How many outliers exist and where they are located, e.g., if they are extreme outliers or if they are relatively close to the whiskers.
- If the median is in the mid of the box or if the median is closer to one of the quartiles. Together with the lengths of each whisker, this can be be used as an indicator for skew in the data.

We can learn a lot about the Iris data from the boxplot, e.g., the following.

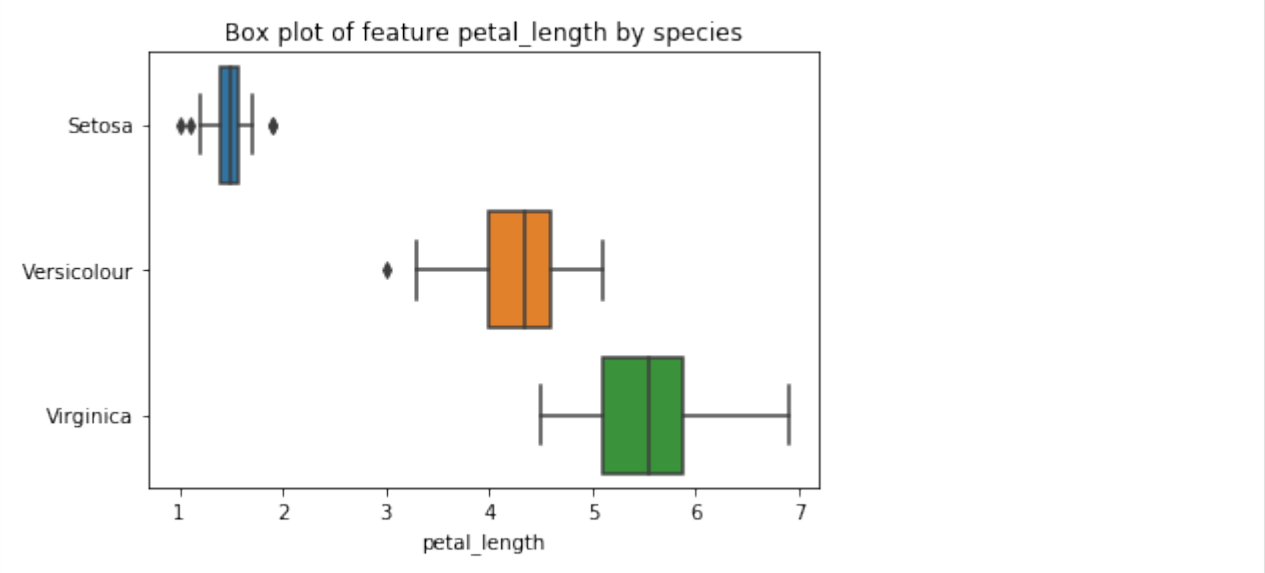
- The features do not have the same ranges. The range of the petal length is about twice the range of the sepal width and the petal width, and also larger than the sepal length.
- There are only few outliers in the data. The only outliers are for the sepal width. These outliers are likely not because they are really far away from the data, but rather due to the low IQR, i.e., because the center of the data is very dense. Consequently, 1.5 times the IQR is only a very short range. That this is indeed the case is even more likely, because the outliers are very close to the whiskers.

As you can see, the information from boxplots is different from the information we get through histograms and density plots. Histograms and density plots tell us about the actual probability distribution of the data. We can get deep insights into single features from them. With boxplots, we also look at single features but the focus is on general

statistic properties. Through a boxplot of multiple features, we can better compare them statistically, e.g., if they have similar ranges, as well as get a general overview of the data in a single figure.

3.3.3 Relationships between Features

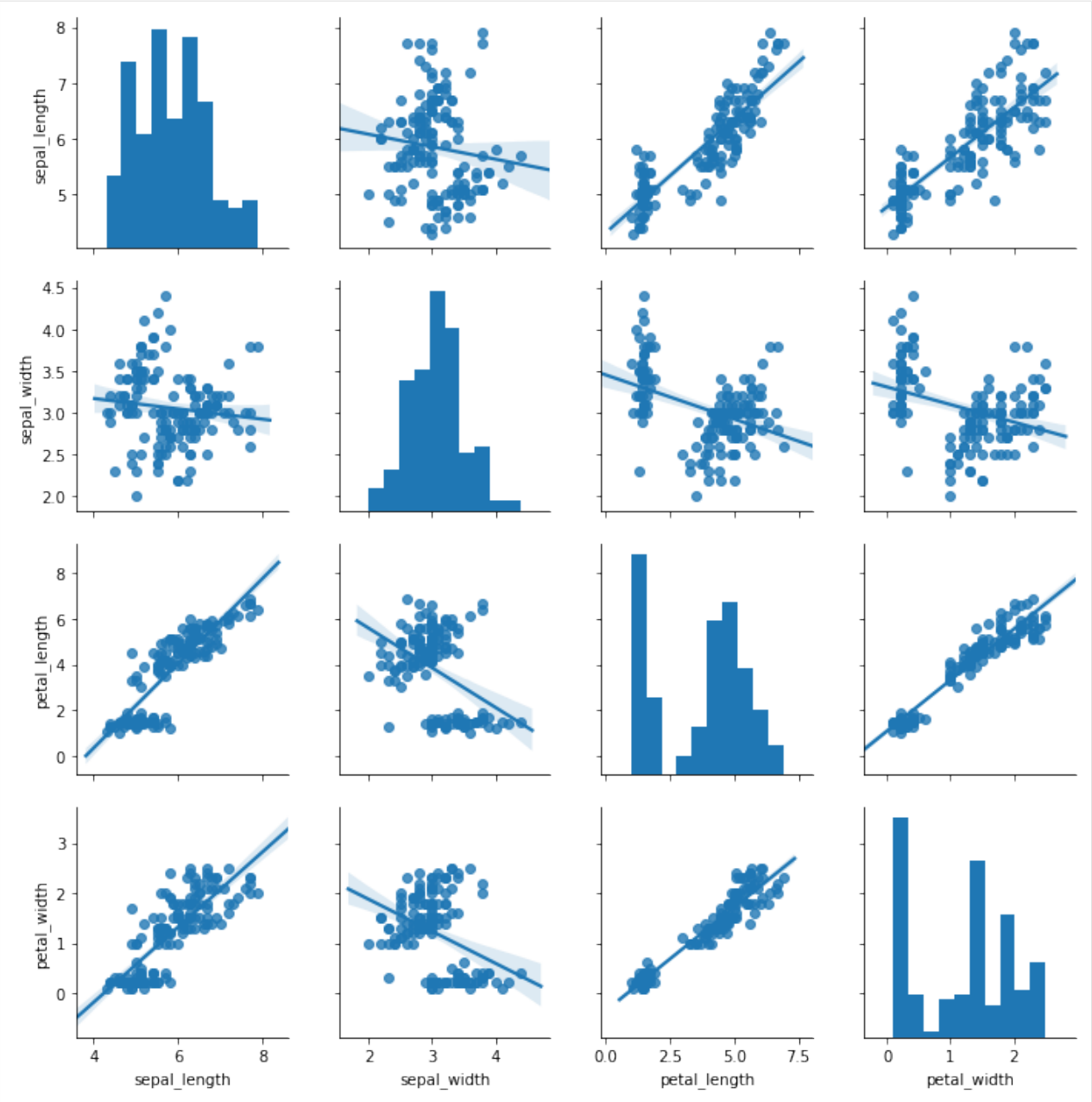
So far, we have analyzed features on their own. With the boxplot, we already looked at multiple features at once, but each box was still only for information from a single feature. Boxplots can, to a limited degree, also be used to get insights into the relationships of features, in case one feature is categorical. For example, we use a boxplot to visualize the petal lengths of the different species of iris.



This boxplot tells us a lot about the relationship between the petal length and the species.

- All setosas have a petal length less than 2.5, all other iris species have petal lengths of at least 2.5.
- Virginicas have a petal length of at least 4.3, hence, all iris with a petal length between 2.5 and 4.3 must be versicolour.
- The petal length of versicolour are all less than 5.1, hence, all iris with a petal length greater than 5.1 must be virginica.
- In the range 4.3 to 5.1 the plans can be either versicolour or virginica.

The drawback of boxplots is that they restrict the comparison to a few statistical markers. Moreover, the relation between features can only be analyzed if one of the features is categorical, ideally also with not too many categories. We can get a more detailed view about the relationship between features with a *pair-wise scatterplot*.



As you can see, the pair-wise scatterplot is not a single plot, but a collection of plots that is organized as a matrix. The diagonal of the matrix are histograms of the features. The other plots show the relationship between two features. Each point in the plots represents one instance. The lines represent how a linear regression would be fit to the data. If the points are close to the line, this means a linear relationship, if they are spread out, the relationship is not linear. If the slope of the line is negative (downwards), the correlation between the data is negative, if the slope is positive (upwards), the correlation is positive. The values on the x-axis are for the feature defined at the very bottom of the visualization, the values on the y-axis are for the feature defined on the very left of the visualization. For example, the second plot in the first row shows the relationship between the sepal width on the x-axis and the sepal length on the y-axis.

From the above plot, we can learn both about the individual features through the histogram, as well as the pairwise relationships between the features. Here are some examples for what we can learn.

- The sepal width seems to follow a normal distribution, the petal length looks like a mixture of two normal

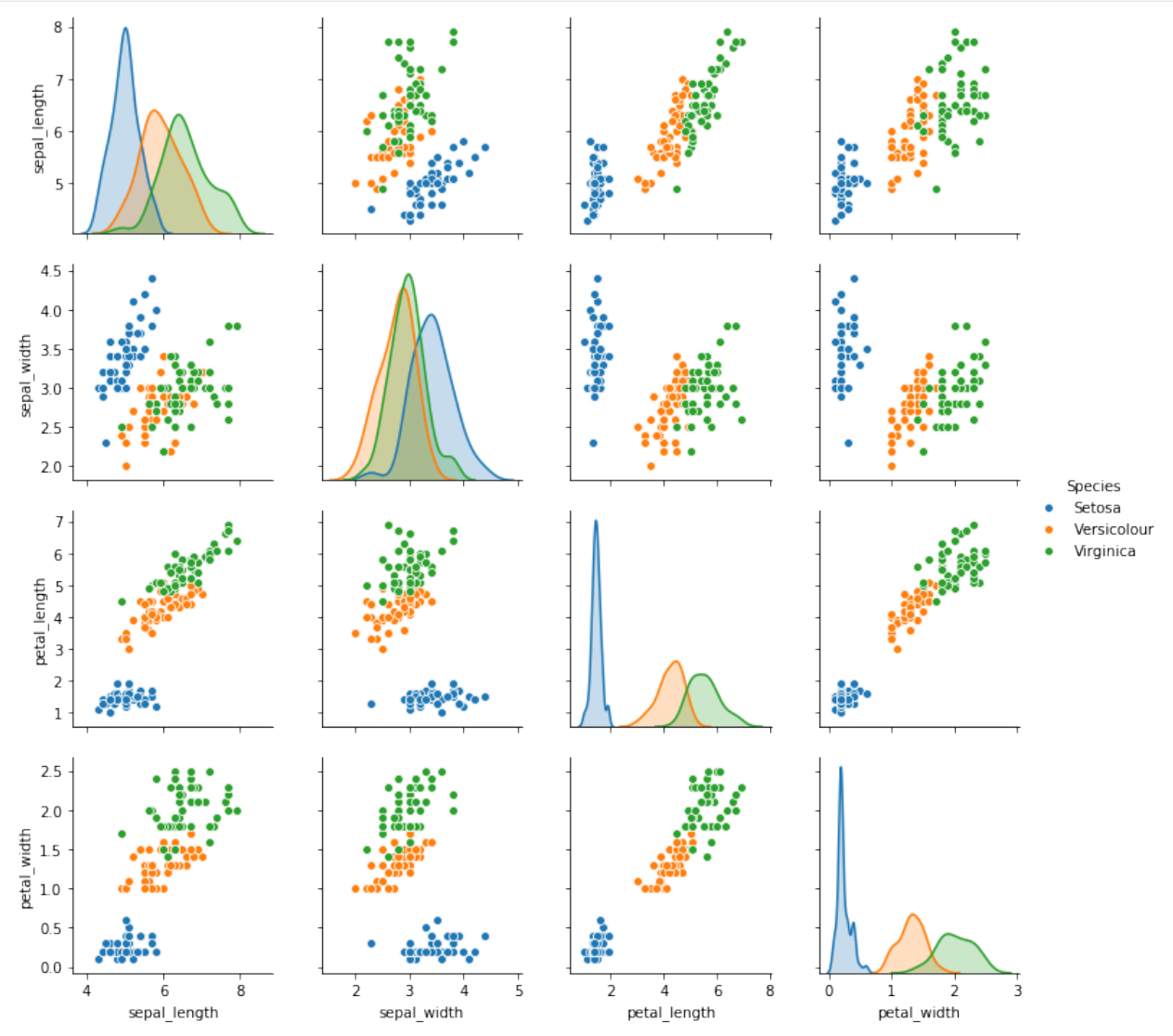
distributions.

- The relationship between sepal width and sepal length looks fairly random. This is indicated because the points a spread about nearly the whole area of the plot.
- The relationship between petal length and petal width seems to be a positive correlation, i.e., if the petal widths increases, the petal length also increases (positive correlation). Moreover, the correlation seems to be linear (data very close to the line), which means that if the petal length increases by, e.g., 10%, the petal width also increases by 10%.
- There are two groups of data that we can clearly see in the plot of the petal length and the petal width: one group with less points is at the bottom left of the plot, another group with more points is at the top-right of the plot. Similar groups can be found at different locations in nearly all plots.

  Note:

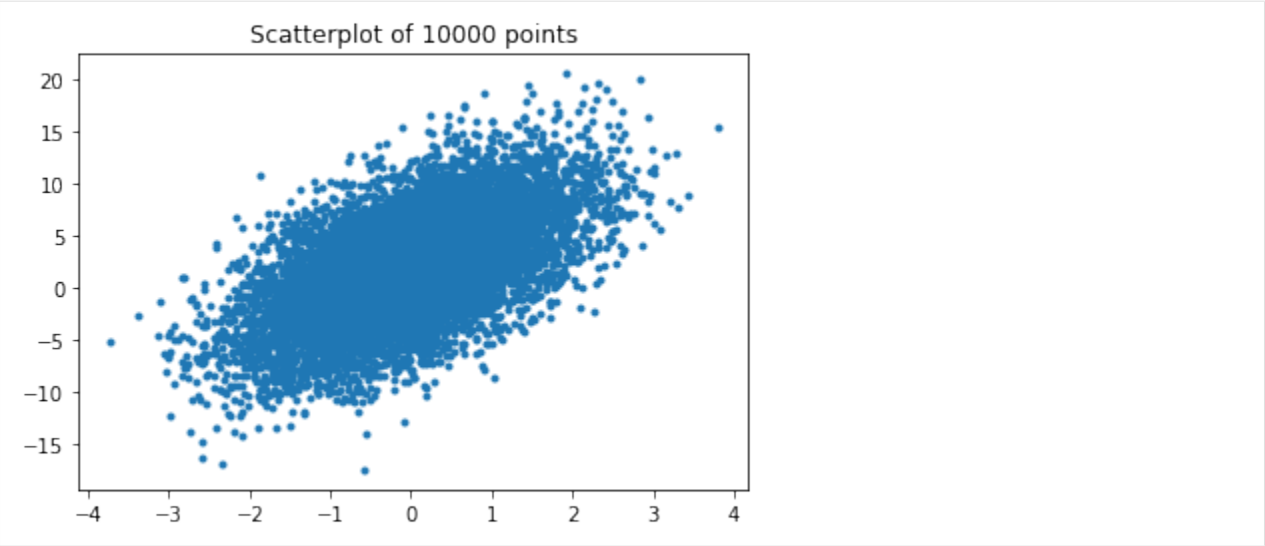
  We do not discuss correlation in detail. The general concept is quite simple. We say that two features X and Y are positively correlated, if X usually increases, if Y increases. The more often this is the case, the stronger the correlation. Vice versa, we say that X and Y are negatively correlated, if X usually decreases, if Y increases. A linear correlation implies that this relationship in the increase is regular such that $X \approx a + b \cdot B$ for two constants $a, b \in \mathbb{R}$. *Correlation coefficients* are measures for the correlation with values in the interval $[-1, 1]$, where 0 is no correlation and the closer the value comes to 1/-1, the stronger the positive/negative correlation.

Categorical variables like the type of iris can be used to enhance the pair-wise plots by encoding the categories as colors. Moreover, we can also show density plots instead of histograms on the diagonal.

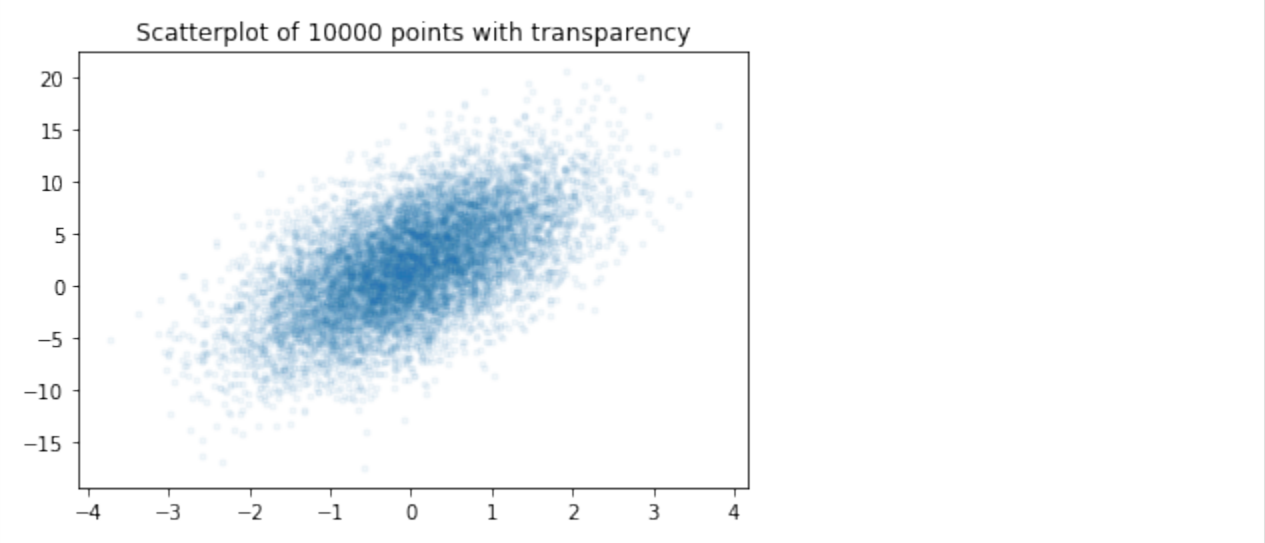


We now see that the group in the bottom left corner is the species setosa. The other group are the versicolour and virginicas and we observe a similar separation between the species, as we did in the boxplot. We also see that it is difficult to separate the species using the data about the sepals, but that an almost perfect separation between the species is possible if the data about petals is used.

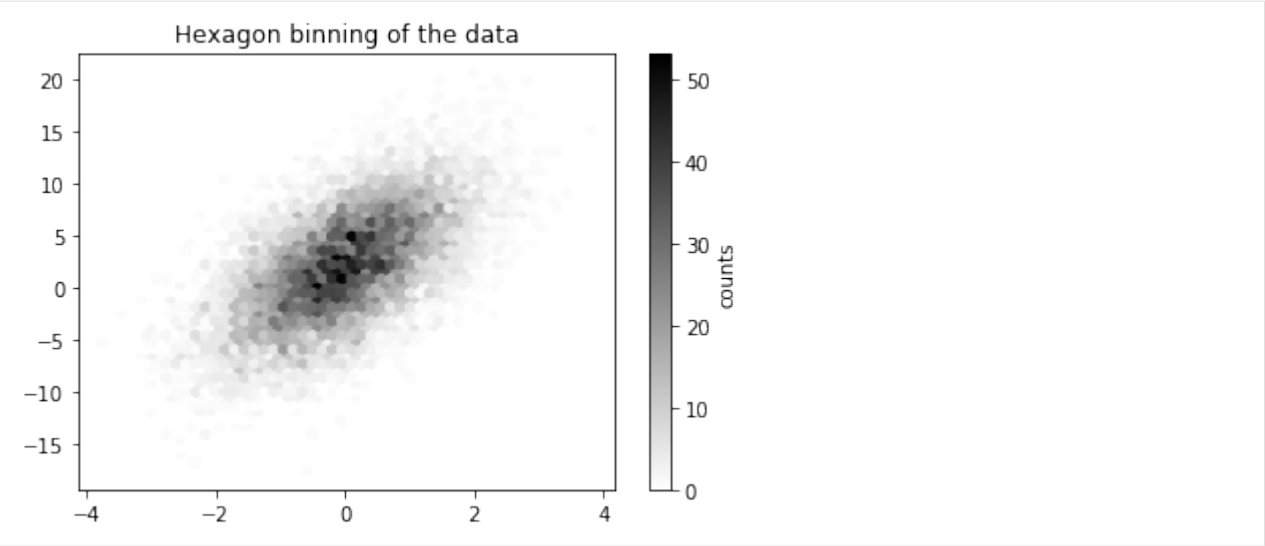
Scatterplots can be problematic, if you have lots of data points in a small area, i.e., *dense* data. When such data is visualized in a scatterplot, we only see one big colored region



The problem with the overlapping data in the dense region is that we do not see the structure of the data anymore. For example, the data in the region could be uniform, normal, a mixture of distributions with multiple spikes, or more or less any other distribution. Thus, we loose a lot of information. There are two ways to deal with this. The simplest way is to just make the points of the scatterplot transparent.

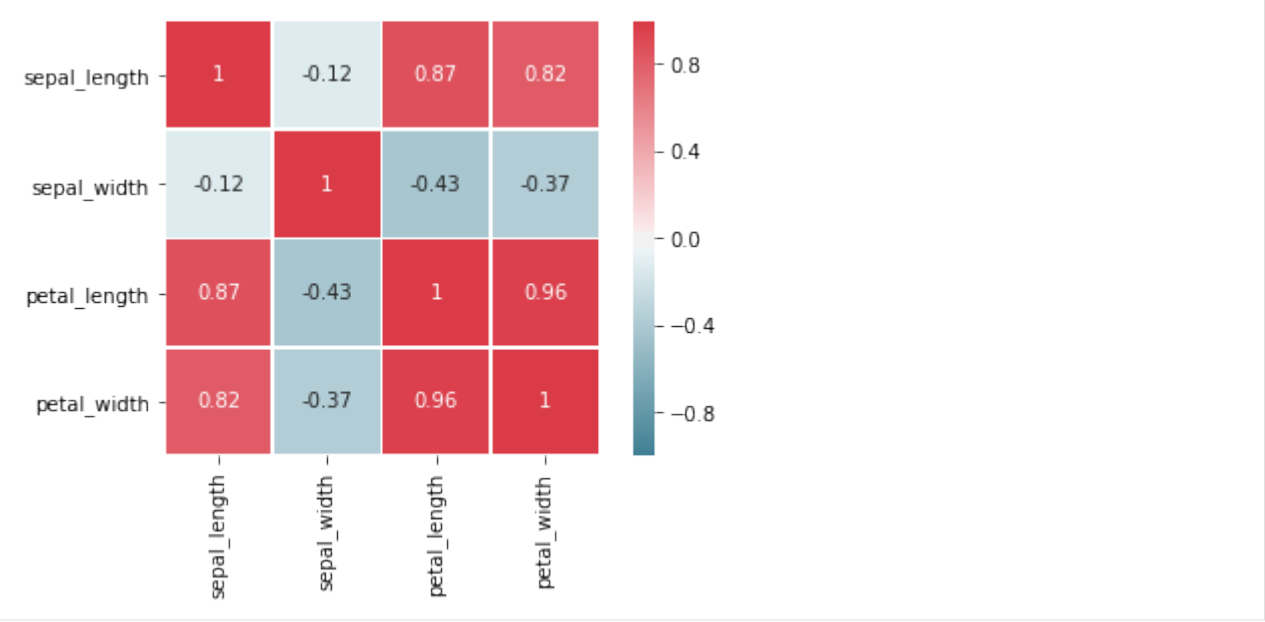


We can now see the structure of the data again and observe that the data is likely normal with the mean value in the the middle of the dense region. The drawback of the transparency solution is that we still do not see how much data is actually located in any region, because we do not have a real color scale. This can be achieved with a hexbin plot.



The hexbin plot is more or less a two-dimensional histogram. The area is split into hexagons (i.e., our bins) and the plot shows using a color scale how much data falls into each hexagon. Now we not only see the structure of the data, but can also say how much data is located in every region of the data.

Before we finish the exploration of the relationship between features, we take another look at correlations. We already briefly discussed that we can observe correlations between features in scatterplots, especially if we add regression lines. However, this approach does not scale well when it comes to the identification of correlations. If we have many features, the number of pair-wise plots we would have to analyze increases quadratically. A more efficient way to get a general overview of correlations is a *correlation heatmap*.



The correlation heatmap visualizes the values of the correlation coefficients between the features. Red means a strong positive correlation, blue means a negative correlation, white means no correlation. The stronger the saturation of the color, the stronger the correlation. The diagonal has a perfect positive correlation, because features are always perfectly correlated with themselves. Moreover, we see that our analysis from the pair-wise plots was correct: there is a very strong positive correlation between the petal length and the petal width. We also see strong negative correlations between the sepal width and the petal length and petal width. Another observation from the pair-wise plots is also confirmed: there is almost no correlation between the sepal length and the sepal width.

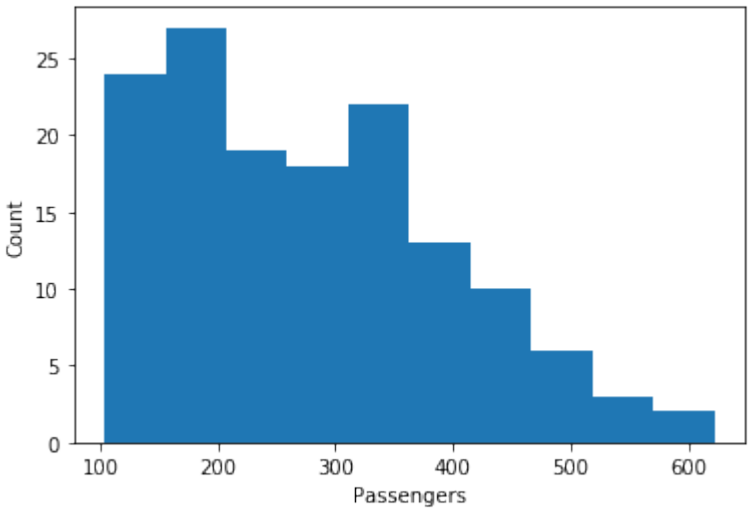
3.3.4 Trends over Time

The visualization approaches so far did not consider that instances may be directly related to each other. The most common relationship is a temporal relationship, e.g., because you have a series of measurements over time. We now use a data set about the number of air passengers of US airlines from 1949 to 1960. The data contains monthly values for the number of passengers.

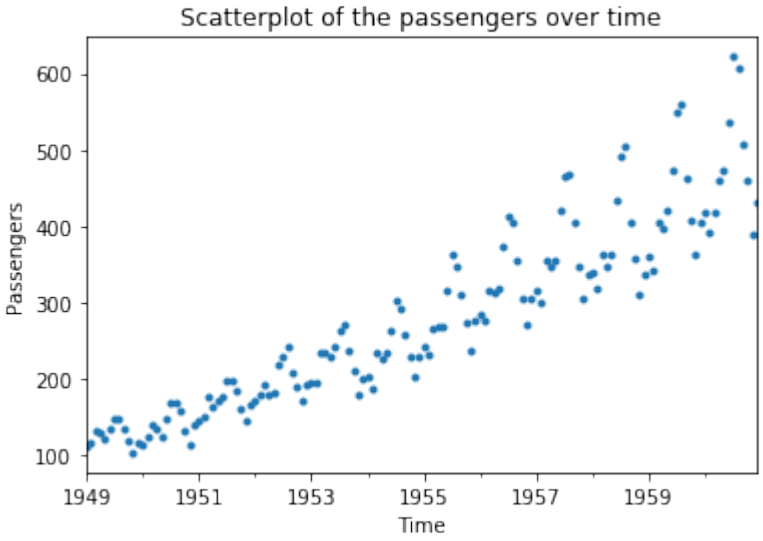
```
!head data/AirPassengers.csv
```

```
Month,#Passengers
1949-01,112
1949-02,118
1949-03,132
1949-04,129
1949-05,121
1949-06,135
1949-07,148
1949-08,148
1949-09,136
```

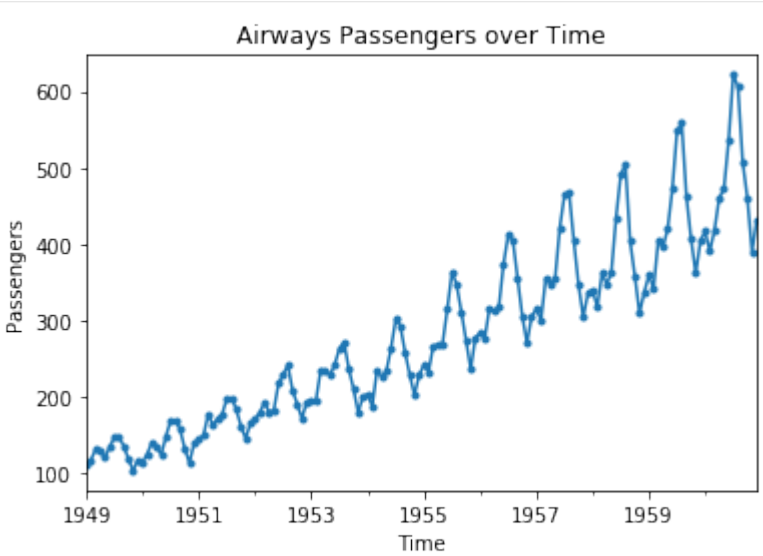
Above, we learned that histograms are a good way to learn about features. Below is a histogram of the number of passengers.



We can learn something from the histogram, e.g., that there where at most around 600 passengers in a month and that there were about 20 months with between 100 and 350 passengers. However, we have no idea how this data relates to the time. We could just say that we get this if we use a scatterplot with the number of passengers as the y-axis and the month as the x-axis.



We get more information from this plot. We now see the general rising trend in the data and also observe that the spread in the trend increases over time. However, we still cannot easily see how data changes from one month to the next. For this, we need to - literally - connect the dots. This is called a line plot.



With the line plot we see the general trend, as well as how the data changes from month to month. Now we also see a regular pattern that repeats every year.

CHAPTER

FOUR

OVERVIEW OF DATA ANALYSIS

4.1 No Free Lunch Theorem (NFL)

Before we dive into the algorithms for the creation of models about the data, we need some foundations. The first foundation is a fundamental theorem about optimization algorithms, the *No Free Lunch Theorem* (NFL). This now gets a bit theoretical, if you are just interested in the (very important!) consequences of the theorem on data modeling, just skip to the colloquial definition. The mathematical formulation of the NFL is the following.

> **No Free Lunch Theorem:**
>
> Let d_m^y be ordered sets of size m with cost values $y \in Y$. Let $f : X \to Y$ a function that should be optimized. Let $P(d_m^y|f, m, a)$ the conditional probability of getting d_m^y by m times running the algorithm a on the function f.
>
> For any pair of algorithms a_1 and a_2 to following equations holds:
>
> $$\sum_f P(d_m^y|f, m, a_1) = \sum_f P(d_m^y|f, m, a_2)$$

A proof for the theorem can be found in the literature. The equation of the NFL tells us the sum of the probabilities of getting certain costs is equal if all functions f are considered for any two algorithms. The meaning of this can be summarized as follows.

> **No Free Lunch Theorem (Colloquial):**
>
> All algorithms are equal when all possible problems are considered.

This is counter-intuitive, because we observe notable differences between different algorithms in practice. However, the NFL does not prohibit that an algorithm is better for some functions f - i.e., for some problems. But if an algorithm is better than the average algorithm for some problems, the NFL tells us that the algorithm must be worse than the average algorithm for other problems. Thus, the true consequence of the NFL is that *there is not one algorithm that is optimal for all problems*. Instead, different algorithms are better for different problems. That is also the reason for the name: there is no "free lunch", i.e., one algorithm for all problems. Instead, data scientists must earn their lunch, i.e., find a suitable algorithm.

This means that having detailed knowledge about one algorithm is not sufficient. Data scientists need a whole toolbox of algorithms. They must be able to select a suitable algorithm, based on the data, their knowledge about the problem, and their experience based on working with different algorithms. Over time, data scientists gather experience with more and more algorithms and become experts in selecting suitable algorithms for a given problem.

4.2 Definition of Machine Learning

Machine learning is currently one of the hottest topics in computer science because recent advances in using the growth of computational power and the available data have led to vast improvements for hard problems, e.g., object recognition, automated translation of languages, and playing games.

To get started with machine learning, we must first understand the meaning of the term. There are many definitions out there. Tom Mitchel gave a very powerful and versatile definition of machine learning:

> **Definition of Machine Learning:**
>
> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E.

At the first glance this definition looks very abstract and the sentence is hard to parse, especially because of the usage of the abstract terms of experience, tasks, and performance measures. However, this is exactly what makes this definition so good: it actually fits the versatile field of machine learning. To understand the definition, we just need to understand what is meant with experience, task, and performance measure.

- The *experience* is (almost always) our data. The more data we have, the more experience we have. However, there are also self-learning systems that do not require data but generate their own experience, like AlphaZero that learns just by playing against itself, i.e., not from data of prior games. In this case, the experience is gained by playing more and more games.
- The *tasks* are what we want to solve. You want to recognize pedestrians so that your self-driving car does not run them over? You want to play and win games? You want to make users click on links of advertisements? Those are your tasks. In general, the task is closely related to your use case and the business objective. In a more abstract way, these tasks can be mapped to generic categories of algorithms, which we discuss later.
- The *performance measure* is an estimate of how good you are at the tasks. Performance measures may be the numbers of pedestrians that are correctly identified and the number of other objects that are mistakenly identified as pedestrians, the number of games that are won, or the number of clicks on advertisements.

With this in mind, the definition is suddenly clear and simple: We just have programs that get better at their tasks with more data.

4.3 Features

Before we start to look at algorithms for machine learning in the next chapters, we first establish some basic concepts. At the foundation of machine learning we have *features*. The meaning of features is best explained using an example. As humans, we immediately know that the following picture shows a whale.

How exactly humans recognize objects is still a matter of research. However, let us assume we identify what different parts of the picture are and then know what the complete picture is about by combining these parts. For example, we might see that there is a roughly *oval shaped* object in the foreground, that is *black on top and white on the bottom*, that *has fins* and is in front of a mostly *blue background*. We can describe the picture using this information as *features*.

This is what machine learning is about: reasoning about objects by representing them as features. Formally, we have an *object space* O with real-world objects and a *feature space* $\mathcal{F}$ with descriptions of the objects. A feature map $\phi : O \to \mathcal{F}$ maps the objects to their representation in the feature space.

For our whale picture, the object space O would be pictures and the feature space $\mathcal{F}$ would have five dimensions: shape, top color, bottom color, background color, has fins. The feature representation of the whale picture would be $\phi(whalepicture) = (oval, black, white, blue, true)$.

There are different types of features, which can be categorized using Steven's levels of measurement into the NOIR schema based on the scale that they are using: nominal, ordinal, interval, and ratios.

Scale	Property	Allowed Operations	Example
Nominal	Classification or membership	$=, \neq$	Color as black, white, and blue
Ordinal	Comparison of levels	$=, \neq, >, <$	Size in small, medium, and large
Interval	Differences of affinities	$=, \neq, >, <, +, -$	Dates, temperatures, discrete numeric values
Ratio	Magnitudes or amounts	$=, \neq, >, <$ $, +, -, *, /$	Size in cm, duration in seconds, continuous numeric values

The nominal and ordinal features are also known as *categorical* features. For nominal features, we only have categories, but cannot establish which category is larger or which categories are closer to each other. For ordinal features, we can establish an order between the categories, but we do not know the magnitude of the differences between the categories. Thus, while we may know that *small* is smaller than *medium* and that *medium* is smaller than *large*, we do not know which difference is larger. This is what we get with interval scales. With features on interval scales we can quantify the size of differences. However, we cannot directly establish by which ratio something changed. For example, we cannot say that the date 2000-01-01 is twice as much as the date 1000-01-01. For this, we need a ratio scale. For example, if we measure the difference in year to the year 0 in the Gregorian calendar, we can say that 2000 years difference is twice as much as 1000 years.

Many machine learning algorithms require that features must be represented numerically. While this is the native representation of features on interval and ratio scales, nominal and ordinal features are not numeric. A naive approach

would be to just assign each category a number, e.g, black is 1, white is 2, blue is 3. This naive approach is very dangerous because it would be possible for algorithms to, e.g., calculate the difference between black and white as 2, which does not make sense at all. A better solution is to use a *one-hot encoding*. The concept behind the one-hot encoding is to expand a nominal/ordinal scale into multiple nominal scales with the values 0 and 1. Each category of the original scale gets its own expanded scale, the value of the new feature is 1, if an instance is of that category, and 0 otherwise.

For example, consider a feature with a nominal scale with values black, white and blue, i.e., $x \in \{black, white, blue\}$. We expand this feature into three new features, i.e., $x^{black}, x^{white}, x^{blue} \in \{0, 1\}$. The values of the new features are:

$$x^{black} = \begin{cases} 1 & \text{if } x = \text{black} \\ 0 & \text{otherwise} \end{cases}$$

$$x^{white} = \begin{cases} 1 & \text{if } x = \text{white} \\ 0 & \text{otherwise} \end{cases}$$

$$x^{blue} = \begin{cases} 1 & \text{if } x = \text{blue} \\ 0 & \text{otherwise} \end{cases}$$

With the approach above, we need n new categories, if the scale of our original feature has n distinct values. In the example above, we have three new features for a scale with three values. It is also possible to use one-hot encoding with $n - 1$ new features. We could just leave out x^{blue} from our example and still always know when the value would have been blue: when it is neither black nor white, i.e., when $x^{black} = x^{white} = 0$. This is especially useful to convert scales with exactly two values. Please note that while one-hot encoding is a useful approach, it does not work well if you have lots of categories, because a lot of new features are introduced. Moreover, the order of ordinal features is lost, i.e., we loose information when one-hot encoding is used for ordinal features.

4.4 Training and Test Data

Data is at the core of data science. For us, data consists of *instances* with *features*. Instances are representations of real-world objects through features. The table below shows an example of data.

shape	top color	bottom color	background color	has fins
oval	black	black	blue	true
rectangle	brown	brown	green	false
...	...	...	...	...

Each line in the table is an instance, each column contains the values of the features of that instance. Often, there is also a property you want to learn, i.e., a *value of interest* that is associated with each instance.

shape	top color	bottom color	background color	has fins	value of interest
oval	black	black	blue	true	whale
rectangle	brown	brown	green	false	bear
...	...	...	...	...	...

Depending on whether this value is known and may be used for the creation of models, we distinguish between *supervised learning* and *unsupervised learning*. Examples for both supervised and unsupervised algorithms are discussed *here*.

Data is often organized in different sets of data. The most frequently used terms are *training data* and *test data*. Training data is used for the creation of models, i.e., for everything from the data exploration, the determination of

suitable analysis approaches, as well as experiments with different models to select the best model. You may use your training data as often as possible and transform the data however you like.

In an ideal world, the test data is only used one time, i.e., once you have finished the model building and want to determine how well your model would perform in a real-world estimation. The model you created is then executed against the test data and you evaluate the performance of that model with the criteria you determined in the discovery phase.

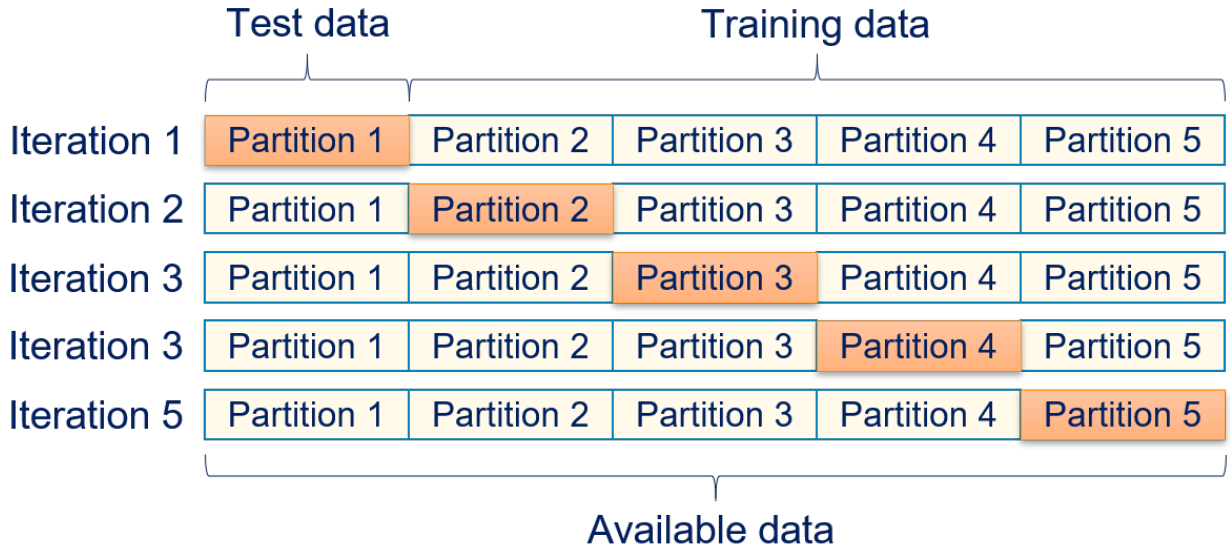
The strong differentiation between training and test data is to prevent *overfitting* and evaluate how your results *generalize*. Overfitting occurs, when data is memorized. For example, we can easily create a model that perfectly predicts our training data: we just memorize all instances and predict their exact value. However, this model would not work at all on other data. Therefore, it would be useless on all other data. This is why we have test data: this data was never seen during the creation of the model. If our model is good and actually does what it is supposed to do, it should also perform well on the test data.

Unfortunately, data is often limited and the creation of more data can be difficult or expensive. On the one hand, we know that machine learning algorithms yield better results with more data. On the other hand, the performance estimate on the test data is also more reliable, if we have more test data. Therefore, how much data is used as training data and how much data is as test data is a tradeoff that must be decided in every project. The best approach is to use *hold-out* data. This is real test data that is only used once. Common ratios between training and test data are, e.g.,

- 50% training data, 50% test data.
- 66.6% training data, 33.4% test data.
- 75% training data, 25% test data.

Sometimes, there are additional concerns to consider. First, there is something called *validation data*. Validation data is "test data for training models". This means that the data is used in the same way the test data is used at the end of the project, but may be used repeatedly during the creation of models. Validation data can, e.g., be used to compare different models to each other and select the best candidate. Validation data can be created as hold-out data as well, e.g., with 50% training data, 25% validation data, and 25% test data.

In case there is not enough data, *cross-validation* may be used to prevent overfitting of models. With cross-validation, there is no clear separation of training and test data. Instead, all data is used for training and testing of the models. With k-fold cross-validation, the data is split into k partitions of equal sizes. Each partition is once used for testing, the other partitions are used for the training.



The performance is then estimated as the mean of the results of the testing on each partition. With cross-validation,

each instance in the data is used as test data exactly once. The drawback of cross-validation is that there is no real test data. Instead, what we called test data - because that is the common terminology in the literature with respect to cross-validation - is actually only validation data. Thus, cross-validation may not detect all types of overfitting and tends to overestimate the performance of models. Thus, you should use hold-out data whenever possible.

4.5 Categories of Algorithms

Depending on the data and the use case, there are different categories of algorithms that may be suitable.

- *Association rule mining* is about finding relationships between items in transactions. The most common use case is the prediction of items, which may be added to a shopping basket, based on items that are already within the basket. We cover the Apriori algorithm for association rule mining in Chapter 5.
- *Clustering* is about finding groups of similar items, respectively the identification of instances that are structurally similar. Both problems are the same because groups of similar items can be inferred by finding structurally similar items. We cover k-means clustering, EM clustering, DBSCAN, and single linkage clustering in Chapter 6.
- *Classification* is about the assignment of labels to objects. We cover the k-nearest neighbor algorithm, decision trees, random forests, logistic regression, naive bayes, support vector machines, and neural networks in Chapter 7.
- *Regression* is about the relationship between features, possibly with the goal to predict numeric values. We cover linear regression with ordinary least squares, ridge, lasso, and the elastic net in Chapter 8.
- *Time series analysis* is about the the analysis of temporal data, e.g., daily, weekly, or monthly data. Time series analysis goes beyond regression, because aspects like the season are considered. We cover ARIMA in Chapter 9.
- *Text mining* of unstructured textual data. All approaches above may be applied to text, once a structure has been imposed. We cover a basic bag-of-words approach for text mining in Chapter 10.

Association rule mining and clustering are examples for unsupervised learning. Both work only based on patterns within the data without any guidance by a value of interest. Hence, such approaches are also called *pattern recognition*. Classification, regression, and time series analysis are examples for supervised learning. These approaches work based on learning how known values can be correctly learned from the values of features.

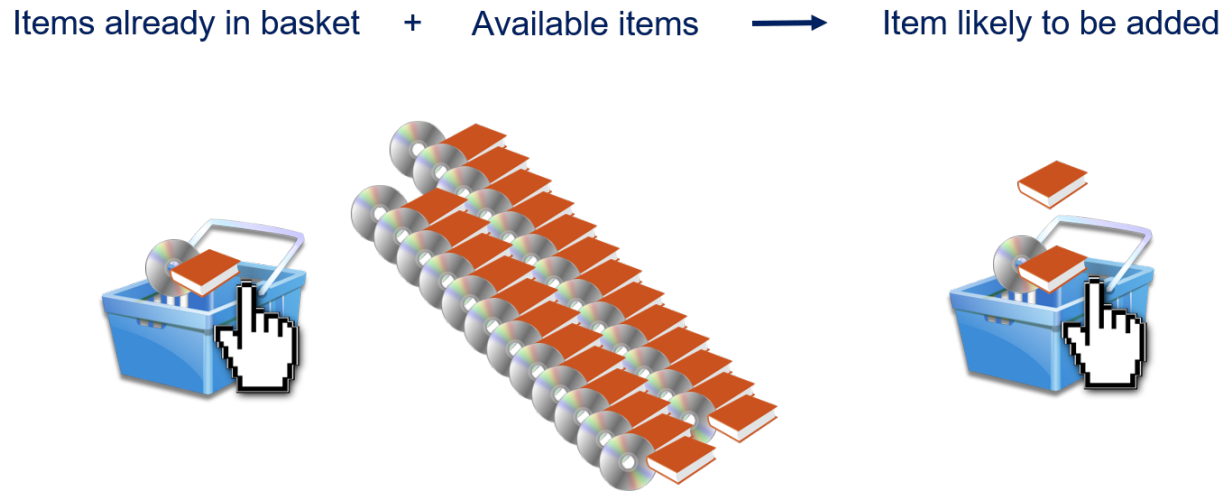
CHAPTER

FIVE

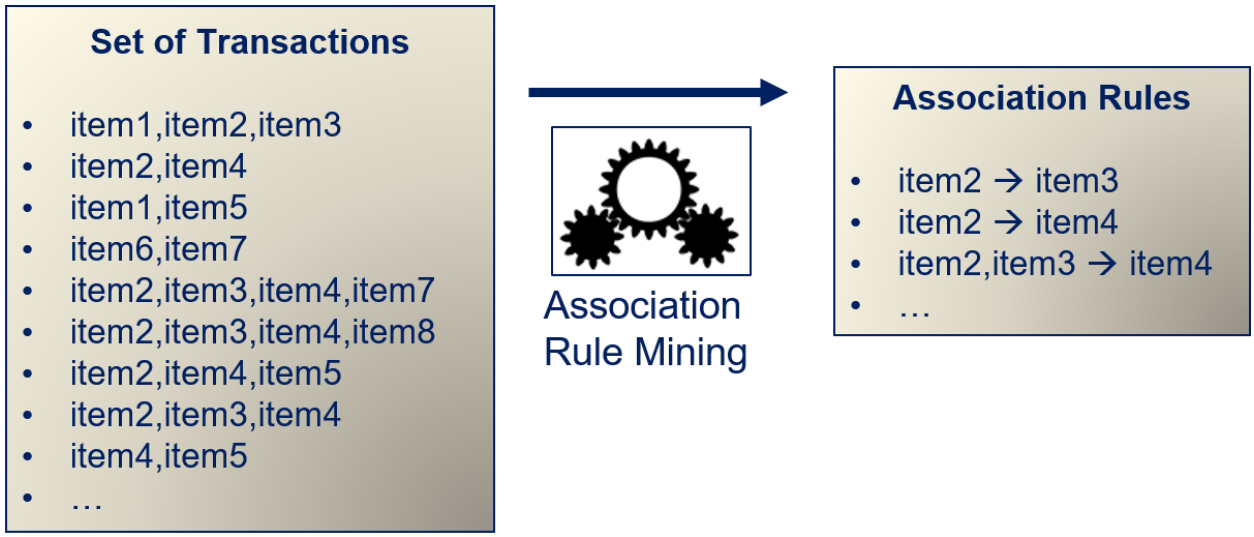
ASSOCIATION RULE MINING

5.1 Overview

Associations are relationships between objects. The idea behind association rule mining is to determine rules, that allow us to identify which objects may be related to a set of objects we already know. In the association rule mining terminology, we refer to the objects as *items*. A common example for association rule mining is basket analysis. A shopper puts items from a store into a basket. Once there are some items in the basket, it is possible to recommend associated items that are available in the store to the shopper.



In this example, the association between items is defined as "shoppers bought items together". More generally speaking, we have *transactions*, and in each transaction we observe a set of related objects. We apply association rule mining to a set of transactions to infer *association rules* that describe the associations between items.



The relationship that the rules describe should be "interesting". The meaning of interesting is defined by the use case. In the example above, interesting is defined as "shoppers bought items together". If the association rules should, e.g., find groups of collaborators, interesting would be defined as "worked together in the past".

We can also formally define association rule mining. We have a finite set of items $I = \{i_1, ..., i_m\}$ and transactions that are a subset of the items, i.e., transactions $T = \{t_1, ..., t_n\}$ with $t_j \subseteq I, j = 1, ..., n$. Association rules are logical rules of the form $X \Rightarrow Y$, where X and Y are disjoint subsets of items, i.e., $X, Y \subseteq I$ and $X \cap Y = \emptyset$. We refer to X as the *antecedent* or left-hand-side of the rule and to Y as the *consequent* or right-hand-side of the rule.

> *Note:*
>
> You may notice that we do not speak of features, but only of items. Moreover, we do not speak of instances, but rather of transactions. The reason for this is two-fold. First, this is the common terminology with respect to association rule mining. Second, there are no real features, the objects are defined by their identity. The transactions are the same as the instances from Chapter 4.

The goal of association rule mining is to identify good rules based on a set of transactions. A generic way to define "interesting relationships" is that items occur often together in transactions. Consider the following example with ten transactions.

```
[['item1', 'item2', 'item3'],
 ['item2', 'item4'],
 ['item1', 'item5'],
 ['item6', 'item7'],
 ['item2', 'item3', 'item4', 'item7'],
 ['item2', 'item3', 'item4', 'item8'],
 ['item2', 'item4', 'item5'],
 ['item2', 'item3', 'item4'],
 ['item4', 'item5'],
 ['item6', 'item7']]
```

We can see that the items item2, item3, and item4 occur often together. Thus, there seems to be an interesting relationship between the items. The question is, how can we find such interesting combinations of items automatically and how can we create good rules from interesting combinations of items.

5.2 The Apriori Algorithm

The Apriori algorithm is a relatively simple, but also powerful algorithm for finding associations.

5.2.1 Support and Frequent Itemsets

The algorithm is based on the concept of the *support* of itemsets $IS \subseteq I$. The support of an itemset IS is defined as the ratio of transaction in which all items $i \in IS$ occur, i.e.,

$$support(IS) = \frac{|\{t \in T : IS \subseteq t\}|}{|T|}.$$

The support mimics our generic definition of interesting from above, because it directly measures how often combinations of items occur. Thus, support is an indirect measure for *interestingness*. What we are still missing is a minimal level of interestingness for us to consider building rules from an itemset. We define this using a minimal level of support that is required for an itemset. All itemsets that have a support greater than this threshold are called *frequent itemset*. Formally, we call an itemset frequent $IS \subset I$, if $support(IS) \geq minsupp$ for a minimal required support $minsupp \in [0, 1]$.

In our example above, the items item2, item3, and item4 would have $support(\{item2, item3, item4\}) = \frac{3}{10}$, because all three items occur together in three of the ten transactions. Thus, if we use $minsupp = 0.3$, this we would call $\{item2, item3, item4\}$ frequent. Overall, we find the following frequent itemsets.

	support	itemsets
0	0.6	(item2)
1	0.4	(item3)
2	0.6	(item4)
3	0.3	(item5)
4	0.3	(item7)
5	0.4	(item2, item3)
6	0.5	(item4, item2)
7	0.3	(item4, item3)
8	0.3	(item4, item2, item3)

5.2.2 Deriving Rules from Itemsets

A frequent itemset is not yet an association rule, i.e., we do not have an antecedent X and a consequent Y to create a rule $X \Rightarrow Y$. There is a simple way to create rules from a frequent itemset. We can just consider all possible splits of the frequent itemset in two partitions, i.e, all combinations $X, Y \subseteq IS$ such that $X \cup Y = IS$ and $X \cap Y = \emptyset$.

This means we can derive eight rules from the itemset $\{item2, item3, item4\}$:

	antecedents	consequents
0	(item4, item2)	(item3)
1	(item4, item3)	(item2)
2	(item2, item3)	(item4)
3	(item4)	(item2, item3)
4	(item2)	(item4, item3)
5	(item3)	(item4, item2)

The two remaining rules use the empty set as antecedent/consequent, i.e., - $\emptyset \Rightarrow \{item2, item3, item4\}$ - $\{item2, item3, item4\} \Rightarrow \emptyset$

Since these rules to not allow for any meaningful associations, we ignore them.

5.2.3 Confidence, Lift, and Leverage

An open question is how we may decide if these rules are good or not. Thus, we need measures to identify if the associations are meaningful or if they are just the result of random effects. This cannot be decided based on the support alone. For example, consider a Web shop with free items. These items are likely in very many baskets. Thus, they will be part of many frequent itemsets. However, we cannot really conclude anything from this item, because it is just added randomly because it is free, not because it is associated with other items. Moreover, the support of a rule $X \Rightarrow Y$ is always the same as for the rule $Y \Rightarrow X$. However, there may be differences, because causal associations are often directed. The measures of *confidence*, *lift*, and *leverage* can be used to see if a rule a is random or if there is really an association.

The confidence of a rule is defined as

$$confidence(X \Rightarrow Y) = \frac{support(X \cup Y)}{support(X)},$$

i.e. the confidence is the ratio of observing the antecedent and the consequent together in relation to only the transactions that contain X. A high confidence indicates that the consequent often occurs when the antecedent is in a transaction.

The lift of a rule is defined as

$$lift(X \Rightarrow Y) = \frac{support(X \cup Y)}{support(X) \cdot support(Y)}.$$

The lift measures the ratio between how often the antecedent and the consequent are observed together and how often they would be expected to be observed together, given their individual support. The denominator is the expected value, given that antecedent and consequent are independent of each other. Thus, a lift of 2 means, that X and Y occur twice as often together, as would be expected if there was no association between the two. If the antecedent and the consequent are completely independent of each other, the lift is 1.

The leverage of a rule is defined as

$$leverage(X \Rightarrow Y) = support(X \cup Y) - support(X) \cdot support(Y).$$

This definition is almost the same as for the lift, except that the difference is used instead of the ratio. Thus, there is a close relationship between lift and leverage. In general, leverage slightly favors itemsets with larger support.

To better understand how the confidence, lift, and leverage work, we look at the values for the rules we derived from the itemset $\{item2, item3, item4\}$.

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage
0	(item4, item2)	(item3)	0.5	0.4	0.3	0.60	1.500000	0.10
1	(item4, item3)	(item2)	0.3	0.6	0.3	1.00	1.666667	0.12
2	(item2, item3)	(item4)	0.4	0.6	0.3	0.75	1.250000	0.06
3	(item4)	(item2, item3)	0.6	0.4	0.3	0.50	1.250000	0.06
4	(item2)	(item4, item3)	0.6	0.3	0.3	0.50	1.666667	0.12
5	(item3)	(item4, item2)	0.4	0.5	0.3	0.75	1.500000	0.10

Based on the metrics, the best rules seem to be $\{item3, item4\} \Rightarrow \{item2\}$. This rule has a perfect confidence of 1, i.e., item2 is always present when item4 and item3 are also present. Moreover, the lift of 1.66 indicates that this is 1.66 times more often than expected. The counterpart to this rule is $\{item2\} \Rightarrow \{item3, item4\}$. The rule has the same lift, but the confidence is only 0.5. This means that item2 appears twice as often alone, as together with the items item3 and item4. Thus, we can estimate that this rule would be wrong about 50% of the time.

The best rule with a single item as antecedent is $\{item3\} \Rightarrow \{item2, item4\}$ with a confidence of 0.75 and a lfit of 1.5. Thus, in 75% of the transactions in which item3 occurs, the items item2 and item4 are also present, which is 1.5 times more often than expected.

The example also shows some general properties of the measures. Most importantly, lift and leverage are the same, if antecedent and consequent are switched, same as the support. Thus, confidence is the only measure we have introduced that takes the causality of the associations into account. We can also observe that the changes in lift and leverage are similar. The lift has the advantage that the values allow a straight forward interpretation.

5.2.4 Exponential Growth

We have now shown how we can determine rules: we need to find frequent itemsets and can then derive rules from these sets. Unfortunately, finding the frequent itemsets is not trivial. The problem is that the number of itemsets grows exponentially with the number of items. The possible itemsets are the powerset $\mathcal{P}$ of I, which means there are $|\mathcal{P}(I)| = 2^{|I|}$ possible itemsets. Obviously, there are only very few use cases, where we would really need to consider all items, because often shorter rules are preferable. Unfortunately, the growth is still exponential, even if we restrict the size. We can use the binomial coefficient

$$\binom{|I|}{k} = \frac{|I|!}{(|I|-k)!k!}$$

to calculate the number of itemsets of size k. For example, if we have |I|=100 items, there are already 161,700 possible itemsets of size $k = 3$. We can generate eight rules for each of these itemsets, thus we already have 1,293,600 possible rules. If we ignore rules with the empty itemset, we still have 970,200 possible rules.

Thus, we need a way to search the possible itemsets strategically to deal with the exponential nature, as attempt to find association rules could easily run out of memory or require massive amounts of computational resources, otherwise.

5.2.5 The Apriori Property

Finally, we come to the reason why this approach is called the Apriori algorithm.

> **Apriori Property**
>
> Let $IS \subseteq I$ be a frequent itemset. All subsets $IS' \subseteq IS$ are also frequent and $support(IS') \geq support(IS)$.

This property allows us to search for frequent itemsets in a bounded way. Instead of calculating all itemsets and then checking if they are frequent, we can *grow* frequent itemsets. Since we know that all subsets of a frequent itemset must be frequent, we know that any itemset that contains a non-frequent subset cannot be frequent. We use this to prune the search space as follows.

1. Start with itemsets of size $k = 1$.
2. Drop all itemsets that do not have the minimal support, so that we only have frequent itemsets left.
3. Create all combinations of the currently known frequent itemsets of size $k + 1$
4. Repeat steps 2 and 3 until

- No frequent itemsets of length $k + 1$ are found
- A threshold for k is reached, i.e., a maximal length of itemsets.

This algorithm can still be exponential. For example, if all transactions contain all items, all possible itemsets are frequent and we still have exponential growth. However, in practice this algorithm scales well, if the support is sufficiently high.

For example, let us consider how we can grow frequent itemsets with $minsupp = 0.3$ for our example data. Here is the data again.

```
[['item1', 'item2', 'item3'],
 ['item2', 'item4'],
 ['item1', 'item5'],
 ['item6', 'item7'],
 ['item2', 'item3', 'item4', 'item7'],
 ['item2', 'item3', 'item4', 'item8'],
 ['item2', 'item4', 'item5'],
 ['item2', 'item3', 'item4'],
 ['item4', 'item5'],
 ['item6', 'item7']]
```

We start by looking at the support of the individual items:

Itemset with $k = 1$	support	Drop
$\{item1\}$	0.2	x
$\{item2\}$	0.6	
$\{item3\}$	0.4	
$\{item4\}$	0.5	
$\{item5\}$	0.3	
$\{item6\}$	0.2	x
$\{item7\}$	0.3	
$\{item8\}$	0.1	x

Since the items item1, item6, and item8 do not have the minimal support, we can drop them and do not need to consider them when we go to the itemsets of size $k = 2$.

Itemset with $k = 2$	support	Drop
$\{item2, item3\}$	0.4	
$\{item2, item4\}$	0.5	
$\{item2, item5\}$	0.1	x
$\{item2, item7\}$	0.1	x
$\{item3, item4\}$	0.3	
$\{item3, item5\}$	0.0	x
$\{item3, item7\}$	0.1	x
$\{item4, item5\}$	0.2	x
$\{item4, item7\}$	0.1	x
$\{item5, item7\}$	0.0	x

As we can see, only the combinations with the items item2, item3, and item4 are frequent, all others can be dropped. This leaves is with a single combination for $k = 3$.

Itemset with $k = 3$	support	Drop
$\{item2, item3, item4\}$	0.3	

There are no combinations of length $k = 4$ possible and we are finished. We only had to evaluate the support for $8 + 10 + 1 = 19$ itemsets to find all frequent itemsets among all possible $2^8 = 256$ itemsets, i.e., we could reduce the effort by about 93% by exploiting the Apriori property and growing the itemsets.

5.2.6 Restrictions for Rules

So far, we always consider all possible combinations of antecedent and consequent as rules, except rules with the empty itemset. Another common restriction is to only consider rules with a single item as consequent. The advantage of such rules is that they have a higher confidence than other combinations. This means that the associations are usually strong and, consequently, more often correct.

5.3 Evaluating Association Rules

The final question that we have not yet answered is how we can determine if the associations rules we determined are good, i.e., if we found real associations and not random rules. The confidence, lift, and leverage already support this and if these measures are used in combination they are a good tool to identify rules. Confidence tells you if the relationship may be random, because the antecedent occurs very often, lift and leverage can tell you if the relationship is coincidental.

However, there are additional ways to validate that the association rules are good. For example, you can split your data into training and test data. You can then evaluate how often the associations you find in the training data also appear in the test data. If there is a big overlap, the association rules are likely good. You can also go one step further and remove items from transaction in the test data and see if your rules can predict the missing items correctly.

Finally, association rule mining is a typical example of a problem where you can achieve decent results with full automation, but likely require manual intervention to achieve very good results. Just think back to a strange recommendation you may have seen in a Web shop at some point. This was likely because there was no manual validation of the rules and the result of strange buying behavior of single customers. This can be further improved, e.g., through manual inspection of rules and filtering the automatically inferred rules. However, this is not a task for the data scientist alone, but should be supported by domain experts. The goal is to determine for the rules if the associations really make sense and only use the valid rules.

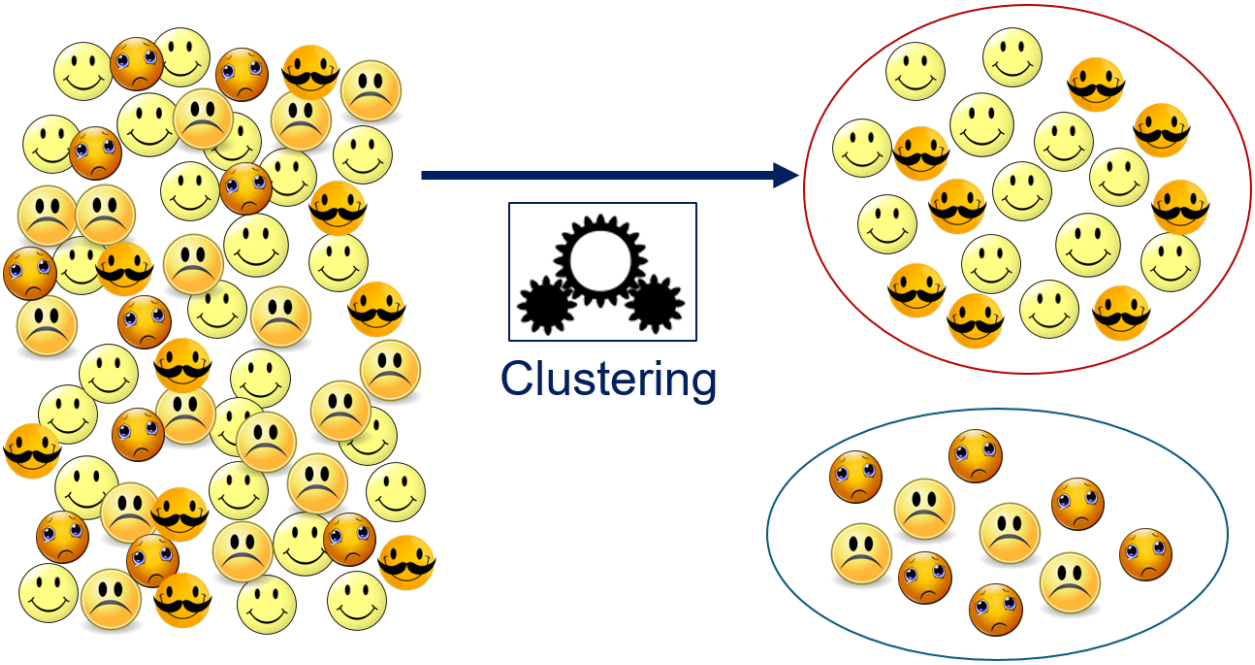
CHAPTER

SIX

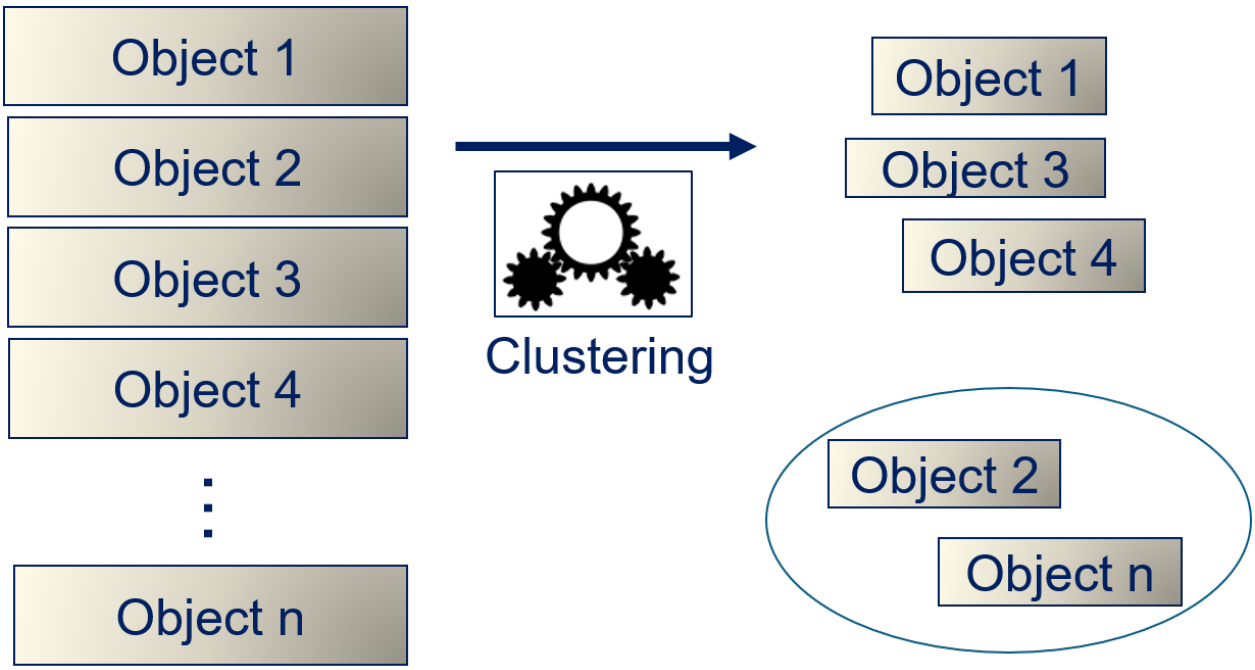
CLUSTERING

6.1 Overview

Clustering is about finding groups of related objects within instances. Consider the following example.



On the left side are a lot of emoticons, which are objects. Clustering takes all these objects and separates them into groups. Above, we have two groups: one with happy emoticons and one with sad emoticons. The groups are created based on the features of the objects, no other information is used. To get a separation into happy and sad emoticons, the features must reflect the emotion. If the features would, e.g., reflect the colors, the clustering algorithm would separate the smilies into yellow and orange emoticons. If the features would reflect the degree of facial hair, the clustering algorithm would separate the emoticons into those with no beard and those with a moustache. A bit more abstract, clustering can be described as follows.



Thus, we just have objects and a clustering algorithm that determines groups of related objects as result. What we do not know without inspection of the clusters is what exactly the relationship between the objects in a group is.

6.1.1 The Formal Problem

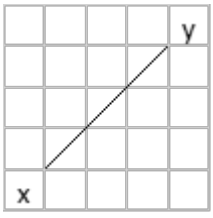
Formally, we have a set of objects $O = \{object_1, object_2, ...\}$ that may be infinite. Moreover, we have representations of these objects in a feature space $\mathcal{F} = \{\phi(o) : o \in O\}$. For clustering, the feature space is usually numeric, i.e., $\mathcal{F} \subseteq \mathbb{R}^d$. The grouping of the objects is described by a map $c : \mathcal{F} \to G$ with $G = \{1, ..., k\}$ and $k \in \mathbb{N}$ the number of clusters.

6.1.2 Measuring Similarity

Clusters are groups of *similar* objects. Consequently, the definition of similarity is central to clustering. The approach for clustering is simple: similar means that there is a short *distance* between objects in the feature space. There are different ways to measure distances, as any well-defined metric can be used. In the following, we present the three most commonly used metrics.

Euclidean Distance

By far the most common distance measure is the *euclidean distance*. The euclidean distance is the direct line between two points.

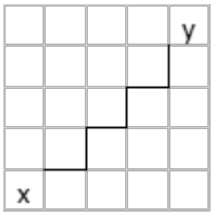


The euclidean distance is based on the euclidean norm $|| \cdot ||_2$ and defined as

$$d(x, y) = ||y - x||_2 = \sqrt{(y_1 - x_1)^2 + ... + (y_n - x_n)^2}.$$

Manhatten Distance

Another common distance measure is the *Manhatten distance*, also known as taxi-cab norm. The name of the norm comes from the grid of the streets in Manhatten, that is mostly organized in an axis parallel way. Thus, if you want to travel between two points, you cannot use the diagonal.



The Manhatten distances is based on the Manhatten norm $|| \cdot ||_1$ and defined as

$$d(x, y) = ||y - x||_1 = |y_1 - x_1| + ... + |y_n - x_n|.$$

Chebyshev Distance

The *Chebyshev distance* is another common distance measure. The Chebyshev distance is also known as maximum metric and Chessboard distance. This is because the Chebychev distance is the same as the number of moves that the King in the game of chess needs to reach a certain field.

2	2	2	2	2
2	1	1	1	2
2	1	0	1	2
2	1	1	1	2
2	2	2	2	2

Thus, the value of the Chebychev norm is maximum of the distances in any direction. The Chebyshev distance is based on the maximum norm $|| \cdot ||_\infty$ and defined as

$$d(x, y) = ||y - x||_\infty = \max_{i=1,...,n} |y_i - x_i|.$$

6.1.3 Evaluation of Clustering Results

The evaluation of clustering results is a bit different from the evaluation of the other categories of algorithms. The reason for this is that there are no generic criteria. Instead, the criteria depend on the clustering algorithm and the use case.

Each clustering algorithm comes with its own approach for finding related objects, i.e., its own concept of similarity. Consequently, how good the clusters match the data, such that the data within the clusters is really similar to each other, depends on the definition of similarity used by the clustering algorithm. We will discuss such criteria together with algorithms.

The importance of the use case can be seen using the emoticon example above. If we are interested in emotion, we have a perfect grouping. If we, on the other hand, want to identify emoticons from different sets of emoticons, the clusters should be the orange ones and the yellow ones. Unfortunately, such aspects cannot be measured by generic

metrics. Instead, the data scientists, possibly together with domain experts, must analyze the meaning of the clusters in depth. Criteria for such a manual analysis may be if the number of clusters is reasonable, if the objects within a cluster seem similar to domain experts, or if all similar objects are within the same cluster. Unfortunately, such a manual analysis can be very difficult, especially if the data is very large, there are many clusters, or the data has many different features and thus a high dimension.

6.1.4 Cities and Houses

We use a running analogy to explain the concepts behind the clustering algorithms. The analogy is quite simple: instead of objects, the analogy uses "houses", and instead of clusters, the analogy uses "cities". The analogy works nicely, because you do not need to know to which city a house belongs and do not even need to know how many cities there are, to see if a house belongs to the same city as other houses. This is the same problem, we try to solve with clustering. We do not know to which cluster objects belong and how many clusters there are, but want to use an algorithm that can infer this.

6.2 k-Means Clustering

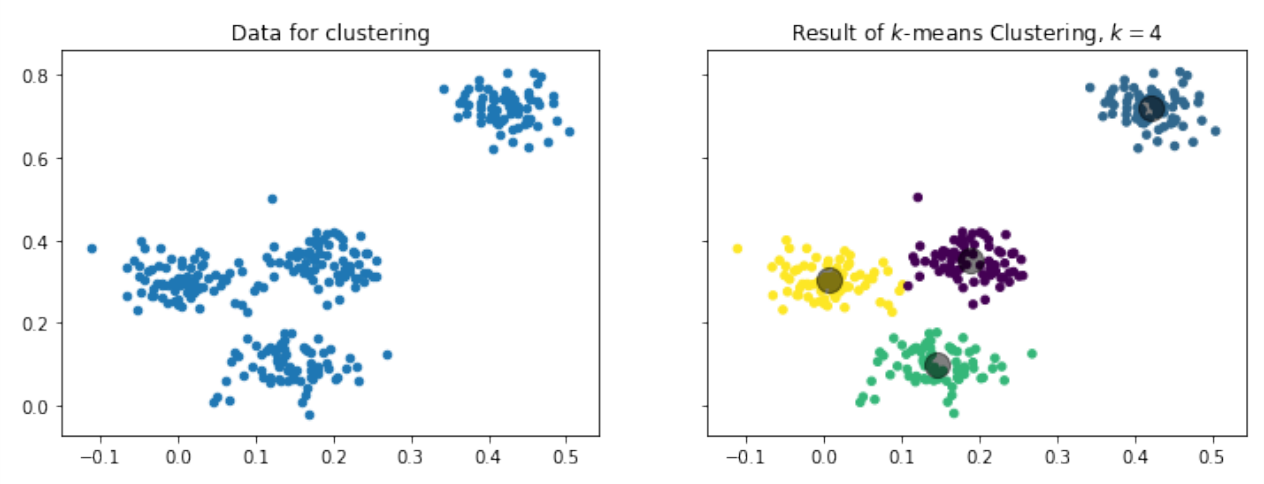
6.2.1 General Idea

When we think about how we can define cities, we may do so by using the town halls as reference points. A simple concept would be to say that each house belongs to the city, whose town hall is closest. This is the essence of of the k-Means algorithm: we describe clusters by their *center*. These centers are also called *centroids* and k-means is an example for *centroid-based clustering*.

If we want to know to which cluster an instance belongs, we just look for the closest center. The k defines the number of clusters. Formally, we have cluster centers $C_1, ..., C_k \in \mathcal{F}$ and a distance metric d. We can determine the cluster of any instance $x \in \mathcal{F}$ as

$$c(x) = argmin_{i=1,...,k} d(x, C_i).$$

The following figure visualizes how data is divided into $k = 4$ clusters. The centers are shown by the large grey points, the colors indicate to which cluster each instance belongs.



This concept is simple, effective, and also intuitive.

6.2.2 The Algorithm

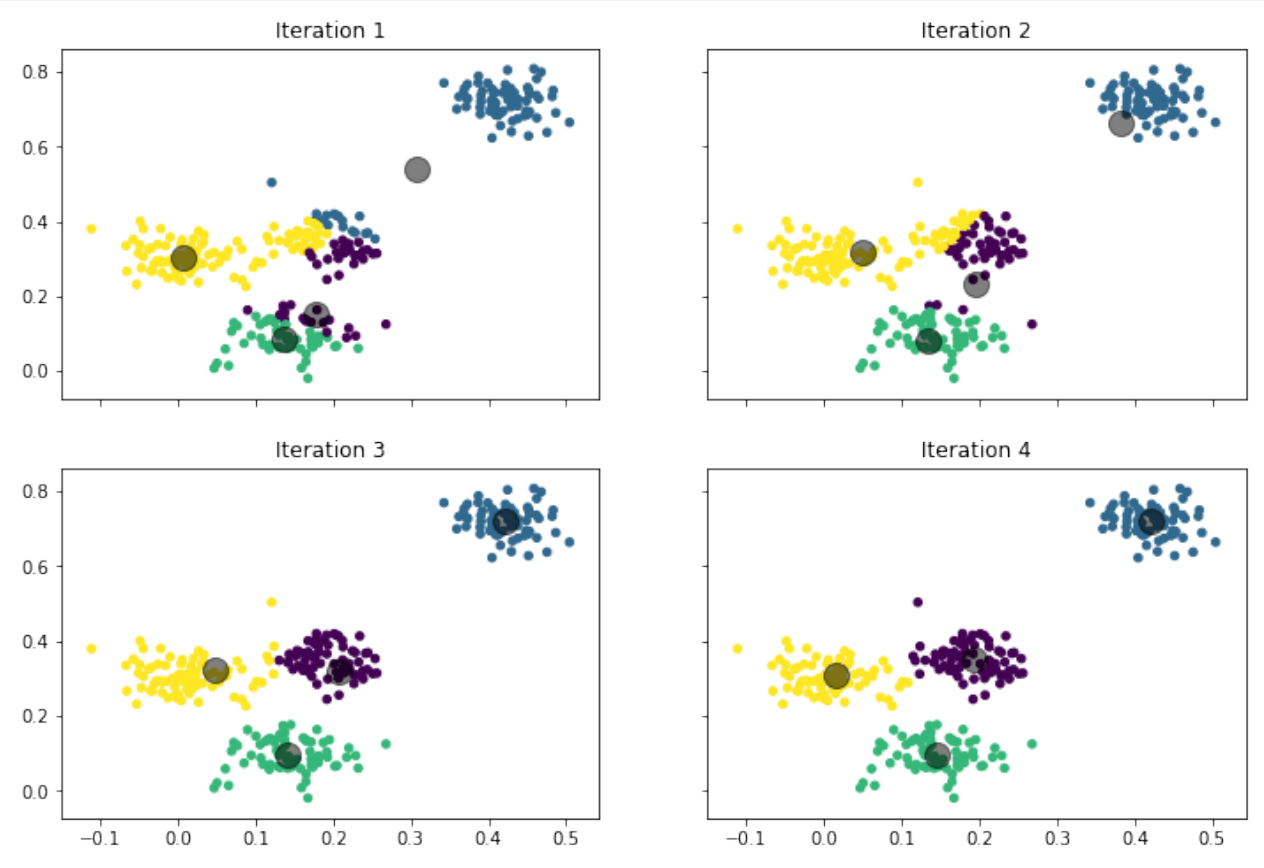
The algorithm to determine the best location for the centers extends this concept. Consider that you have a city. The "best" location for the town hall would be such that the distances between the houses of the city and the town hall is minimized. Now imagine that a couple of new houses are built in the town, while some older ones are destroyed. This means that the location of the town hall should be changed, such that it minimizes the distances again. Of course, this does not happen in real-life, but the concept can be used to define the algorithm to determine the clusters with the k-means algorithm.

Let $X \subset \mathcal{F}$ be our data. We can determine clusters with k-Means as follows.

1. Select initial centroids $C_1, ..., C_k \in \mathcal{F}$.
2. Determine the clusters $X_i = \{x \in X : c(x) = i\}$ for $i = 1, ..., k$.
3. Update centroid such that the location of the centroid is the arithmetic mean of the objects assigned to the clusters such that $C_i = \frac{1}{|X_i|} \sum_{x \in X_i} x$.
4. Repeat steps 2 and 3 until

- Convergence, i.e., the clusters $X_i, i = 1, ..., k$ do no change anymore.
- A predefined maximum number of iterations is reached.

While this may seem abstract, the concept becomes clear when we look at the visualization below.



This is what happens:

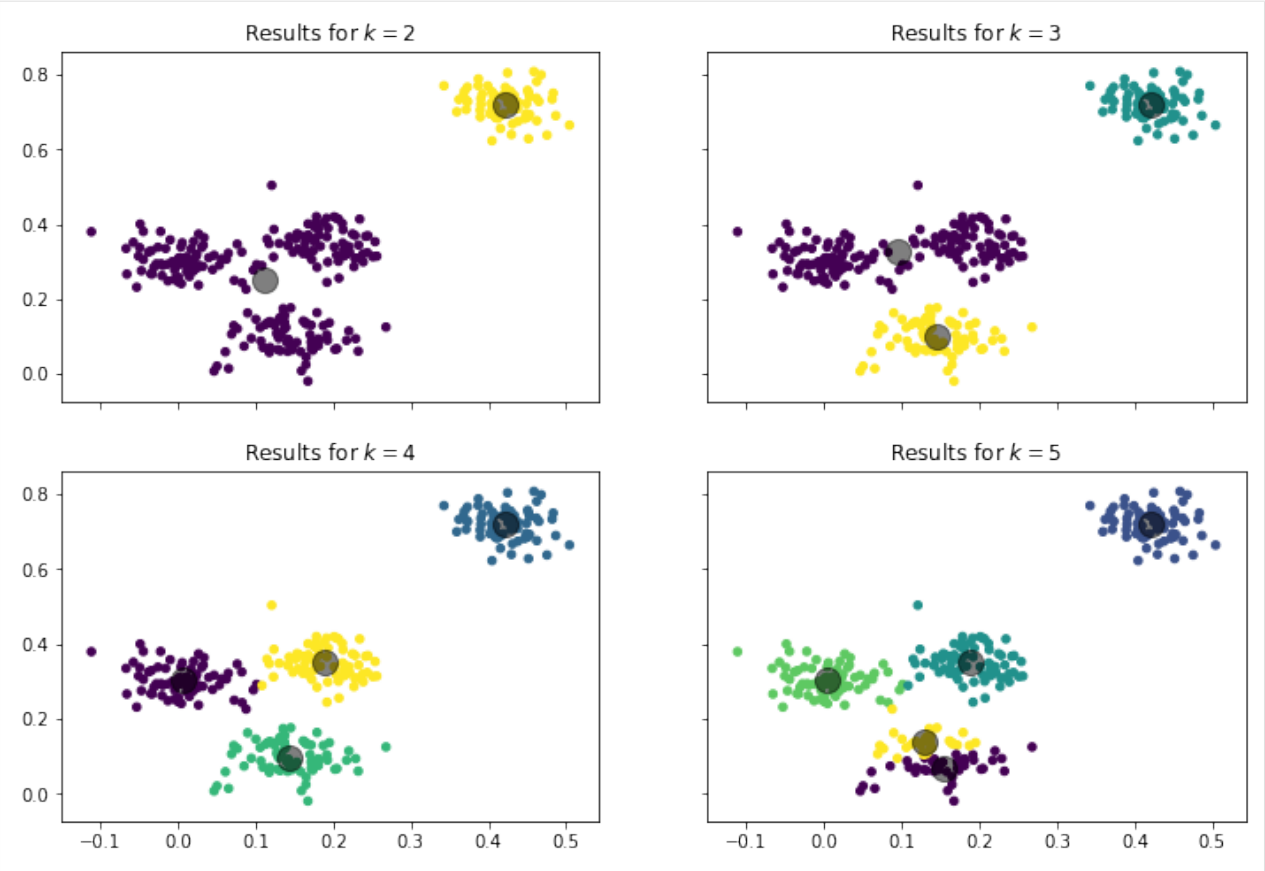
- Iteration 1: Iteration 1 shows the starting position. We start with random cluster locations. One center is actually far away from any data point. The color indicates the initial cluster assignment. We can see that the top-right center gets all the instances in the top-right corner and some instances in the middle (teal). The leftmost center

gets the points on the left side and some from the center (yellow). At the bottom-middle, we have two clusters close to each other which split the points at the bottom (purple and green). Purple also get some points in from the middle.

- Iteration 2: We update the cluster centers and they are moved to the location of the arithmetic mean of the cluster assignments in iteration 1. This means that the teal center moves to the top-right corner, because most points assigned to the cluster are from that area, only some points are from the middle. The teal center is still not located in the middle of the top-right cluster, because of the influence of the points from the middle on the update of the location. However, due to this update, no points from the middle are assigned to the teal cluster anymore. The interaction between yellow, purple, and green is quite interesting. The yellow center moves slightly to the right, because many points from the middle were assigned. The purple cluster moves a bit to the top, because of the points from the middle. The green center stays almost where it was. However, while yellow and purple both moved towards the middle, yellow actually has fewer points assigned, because the move of purple towards the middle was larger. Moreover, while green did not change, most of the bottom cluster is now green, because purple moved to the center.
- Iteration 3: The result begins to converge. Teal is now in the center of the top-right cluster. Yellow starts to move towards the left side, because purple dominates the instances in the middle. As a consequence, green gains full control over the bottom cluster.
- Iteration 4: After only four update cycles, we have almost the final result we have shown above. Teal, purple, and green do not move a lot anymore. Yellow is still moving to the left, but has now also almost reached the final destination.

6.2.3 Selecting k

In the example above, we always used $k = 4$, i.e., four clusters. This is an imporant aspect of the k-Means algorithm: how to pick a suitable value for k? Before we explore this in greater detail, let us look at how the result with different values for k for our sample data.



With $k = 2$ we get a single large cluster and one smaller cluster. With $k = 3$ the large cluster is split in two, and we have one cluster at the top-right, one at the bottom, and one in the middle of the plot. With $k = 4$ the middle cluster is further split into two and we have one in the middle of the plot and one on the right side. With $k = 5$ the bottom cluster is also split in two and now has a top and a bottom half. We can argue that the results for $k = 2, 3, 4$ all make sense. We could argue that $k = 2$ is best, because there is a large gap between the instances on the top-left and the others, and there is no other gap that large. We could also say that $k = 3$ is best, because there is a small gap between the bottom cluster and the points in the middle, and there is no such gap between the points in the middle. We could also say that $k = 4$ is best, because here all clusters are roughly equally dense and have a similar shape. However, for $k = 5$, we cannot really argue that this would be best. The split of the group at the bottom just does not make sense.

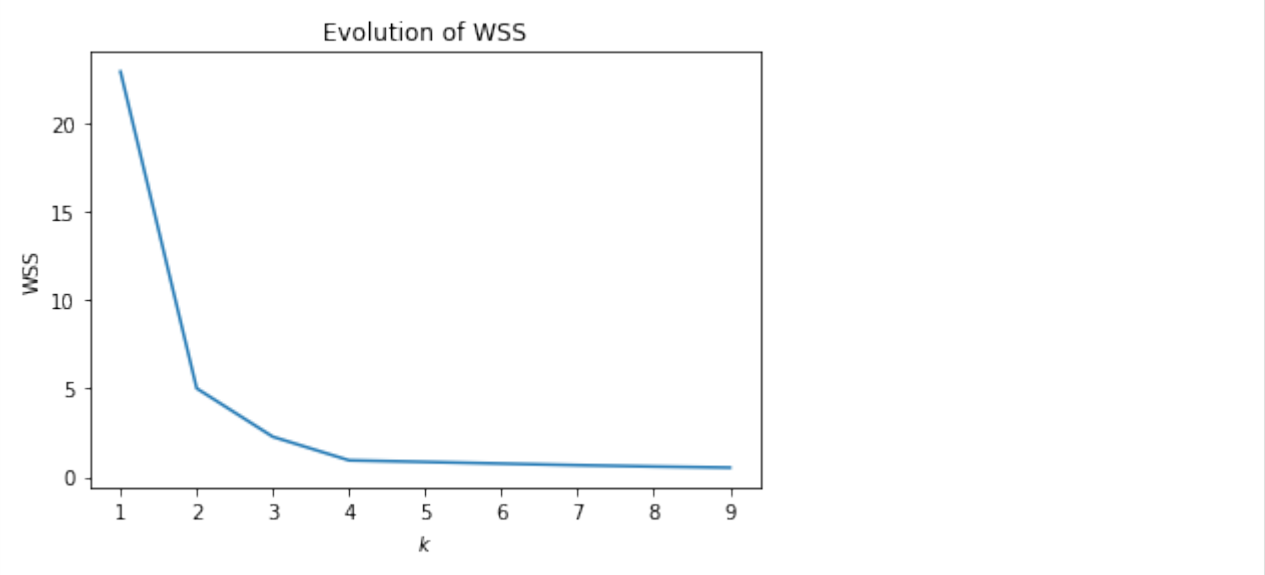
This is a general observation for clustering: without knowing the meaning of instances, we cannot really say which result is best, because often multiple results may make sense. Which is best depends on the use case and the data. In general, there are several aspects that influence the selection of k.

- The domain knowledge about the use case and the data. Knowledge about the domain can help to determine data within a cluster is really similar and also if different clusters are really not similar to each other. If you determine based on domain knowledge that clusters are not internally similar, you need to increase k. If you determine that similar objects are in different clusters, you should decrease k. The use case may also prescribe a fixed number of clusters that must be the outcome. For example, if a binary separation of the data into two groups must be achieved, k must be set to two.
- Visualizations are a great tool to see how clear the separation, how big the gaps between clusters are, and how the data in the clusters is spread out. Visualization is the means for identifying good values of k we used above.
- There is also a more analytical approach for the selection of k based on the *Within-Sum-of-Squares* (WSS). To understand WSS, we must revisit the algorithm of k-Means. The update step sets the centers such that they are located at the arithmetic mean of the cluster points. Thus, the update step minimizes the *intra-cluster variance*. The variance is the square of the standard deviation and calculated by the dividing the sum of the squared

distances from the arithmetic mean, divided by the number of instances in the cluster. Since the arithmetic mean is our center, the variance of the instances in a cluster is the sum of the squared distances from the center, divided by the number of instances in the cluster. Thus, the update step of k-means actually minimizes the *sum of squared* distances *within* a cluster. Based on this, WSS more or less measures how good k-means is at minimizing the intra-cluster variance by computing the squared distances of each instance from the respective cluster center:

$$WSS = \sum_{i=1}^{k} \sum_{x \in X_i} d(x, C_i)^2$$

We already discussed how good the different values of k are based on visualization. There is no domain knowledge we could use for randomly generated data. What is missing is an evaluation of the WSS of the example. The plot below shows the WSS for $k = 2, ..., 5$.



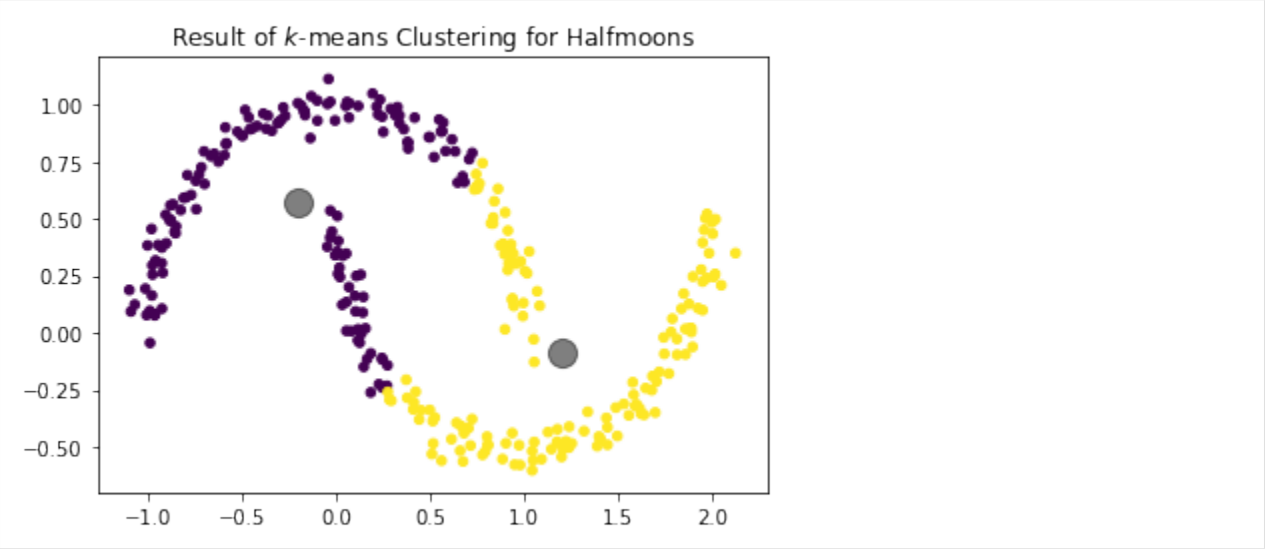
We can see that there is strong decrease in the WSS from 2 to 3 and 3 to 4. There is almost no decrease for values larger than 4. We see that the slope of the WSS changes at both 2, 3, 4. Such changes in slope are called *elbows*, because they look a bit like an arm bent at the elbow. Elbows are the interesting values of k. Thus, from the WSS the values $k = 2, 3, 4$ would all be reasonable. For $k > 4$ the WSS does not decrease a lot anymore and there are no elbows which means that there is no improvements and these are not good values.

The WSS is monotonic-decreasing in k. This means that $WSS(k + 1) \leq WSS(k)$ for all $k > 1$. Consequently, we cannot just pick the minimum, as this would just be the number of instances, i.e., $k = |X|$. The reason for this is, that the variance is always decreasing if we add more centers. This is minimal if $d(x, C_{c(x)}) = 0$ for all $x \in X$, which can only be achieved with |X| clusters, assuming that there are no duplicate values in X. However, if the variance is already small, the decrease will be small, possibly even zero. This is what we observe for $k > 4$ in our example. Thus, we can easily determine reasonable values of k visually, but not automatically by minimizing WSS.

6.2.4 Problems of k-Means

Even though the concept of k-Means is simple, the algorithm often works well. However, there are several problems with k-Means that you should be aware of.

- k-Means is sensitive to the initial clusters. The results may not be stable and change with different start centers. Consequently, k-Means should be run with multiple randomized start centers. If the results are unstable, this is a strong indicator that the outcome may not be good for the given number of clusters k.
- An unsuitable value of k may lead to bad results. Since k must be picked manually, this means that experience is important when using the k-Means algorithm, to avoid bad results due to a wrong number of clusters.
- All features must have a similar range of values, ideally even the same range of values. Otherwise, differences in the ranges of the scales may introduce artificial weights between features. For example, consider data with two features: age in years and the yearly income in Euro. The age is (roughly) between 0 and 100, the income in Euro is - even for low paying jobs, already at 12*400=4,800 Euro, but can also easily be above 50,000 Euro. This means that, when we calculate the distance between two instances, the age is irrelevant, all that matters is the income. Thus, the large range of the income scale means that the income has a higher weight and a stronger influence on the results.
- Because the cluster assignment is based on the distance, clusters tend to be round. Clusters that are not roughly round can often not be described well by centroids. An example are the half-moons below. We can clearly see that there are two half-circles and each one is a cluster. k-Means cannot find these clusters, because they are not round.



6.3 EM Clustering

6.3.1 General Idea

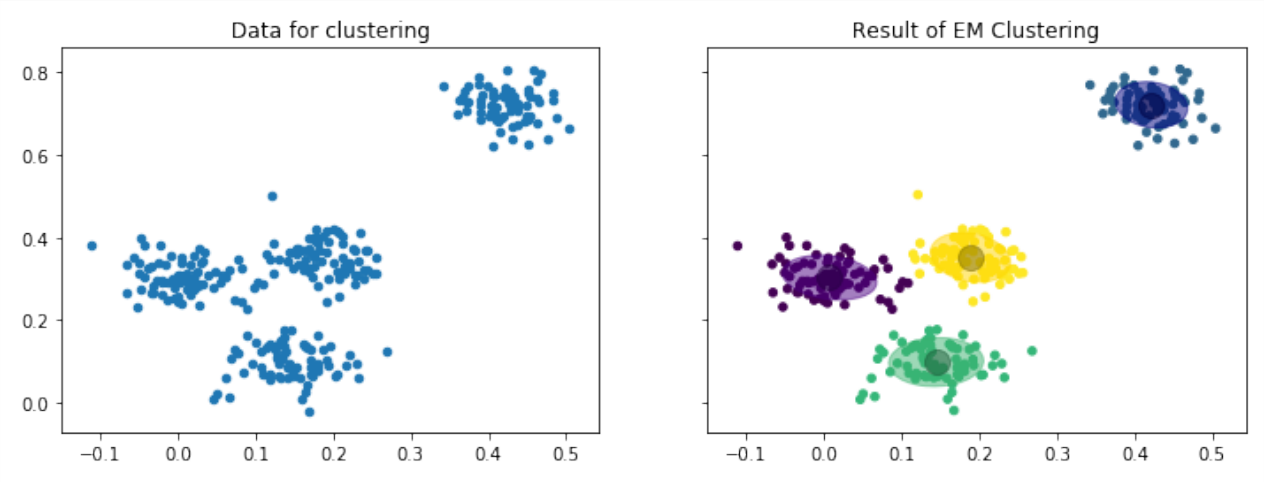
Another way to look at cities and houses is to think of them as random variables. The assignment of houses to cities is then guided by probability distributions. This is an extension of the idea behind k-Means. Our town hall is still the same, i.e., the center. However, we acknowledge that houses closer to the town hall are more likely to belong the the city than houses that are farther away. We actually go one step further and define cities not only based on the location of the town hall, but also by the variability of the houses, i.e., how spread out the houses of the cities are. This concept is called *distribution-based clustering*.

Formally, we have a number of clusters k that are described by random variables $C_1, ..., C_k$. We can calculate the probability that an instance belongs to a cluster $P(C_i = x), i = 1, ..., k$. Each instance is assigned to the *most likely* cluster, i.e., the cluster C_i where the $P(C_i = x)$ is maximized. Consequently, our clustering is defined as

$$c(x) = \max_{i=1,...,k} P(C_i = x).$$

While this concept generally works with any probability distribution, we only use normally distributed random variables to describe the clusters in the following. This is also known as *Gaussian Mixture Model*. Please note that these normal distributions are multivariate, i.e., not just for one dimension, but for as many dimensions as we have features. A univariate normal distribution is described by the mean value and the standard deviation. In comparison, a multivariate normal distribution is described by a vector of mean values and a *covariance matrix*. The covariances describe the relationship between the variances in the different dimensions. A bit simplified, you can think of the covariances as the stretching of the bell-shape of the normal distribution in the different directions. Mathematically, the covariances describe an ellipse. One aspect that is important for us that the covariance matrix is square matrix, where the upper right and lower left triangle are symmetric, i.e., there are $\frac{d \cdot (d+1)}{2}$ free parameters for the covariance matrix. Thus, if we have d features, there are $d + \frac{d \cdot (d+1)}{2}$ parameters that describe the multivariate normal distribution, d mean values and a $d \times d$ covariance matrix. This means that we have $k \cdot (d + \frac{d \cdot (d+1)}{2})$ parameters when we want to describe k clusters.

The following figure shows an example for the result of EM clustering with $k = 4$ normal distributions fit to the same data we already used to show how k-Means works. The mean values are shown by the large gray points, the covariances by the colored ellipses.



While the concept of clustering with random variables as clusters is a bit more complex, the figure already shows the advantage: because we do not just have the mean value, but also the covariances, the clusters also describe how they are *spread*. Thus, we have more information about the data encoded in the cluster.

6.3.2 The Algorithm

This type of clustering is called EM clustering because of the algorithm used to determine the parameters of the clusters is called the *Expectation-Maximization algorithm*. The algorithm is in general similar to the k-Means algorithm we already know: we start with randomly initialized clusters and then iteratively update the clusters to better match the data. For the EM algorithm, the initialization means random mean values and random covariance matrices. Updates mean that the mean values and covariances are updated such that the *likelihood* that the data can be explained by the clusters is increased. We do not discuss the complete mathematical description of the EM algorithm, but rather look at a simplified version, where we update only the mean values and ignore the updates of the covariances.

1. Randomly select initial normal distributions $C_1 \sim (\mu_1, \Sigma_1), ..., C_k \sim (\mu_k, \Sigma_k)$ with $\mu_i \in \mathcal{F}$ the mean values and $\Sigma_i \in \mathcal{F} \times \mathcal{F}$ the covariance matrices.

2. Expectation Step: Determine weights

$$w_i(x) = \frac{p(x|\mu_i, \Sigma_i)}{\sum_{j=1}^{k} p(x|\mu_j, \Sigma_j)}$$

   for all $x \in X$ and $i = 1, ..., k$ as the index of the clusters.

3. Maximization Step: Update mean values as

$$\mu_i = \frac{1}{|X|} \sum_{x \in X} w_i(x) \cdot x$$

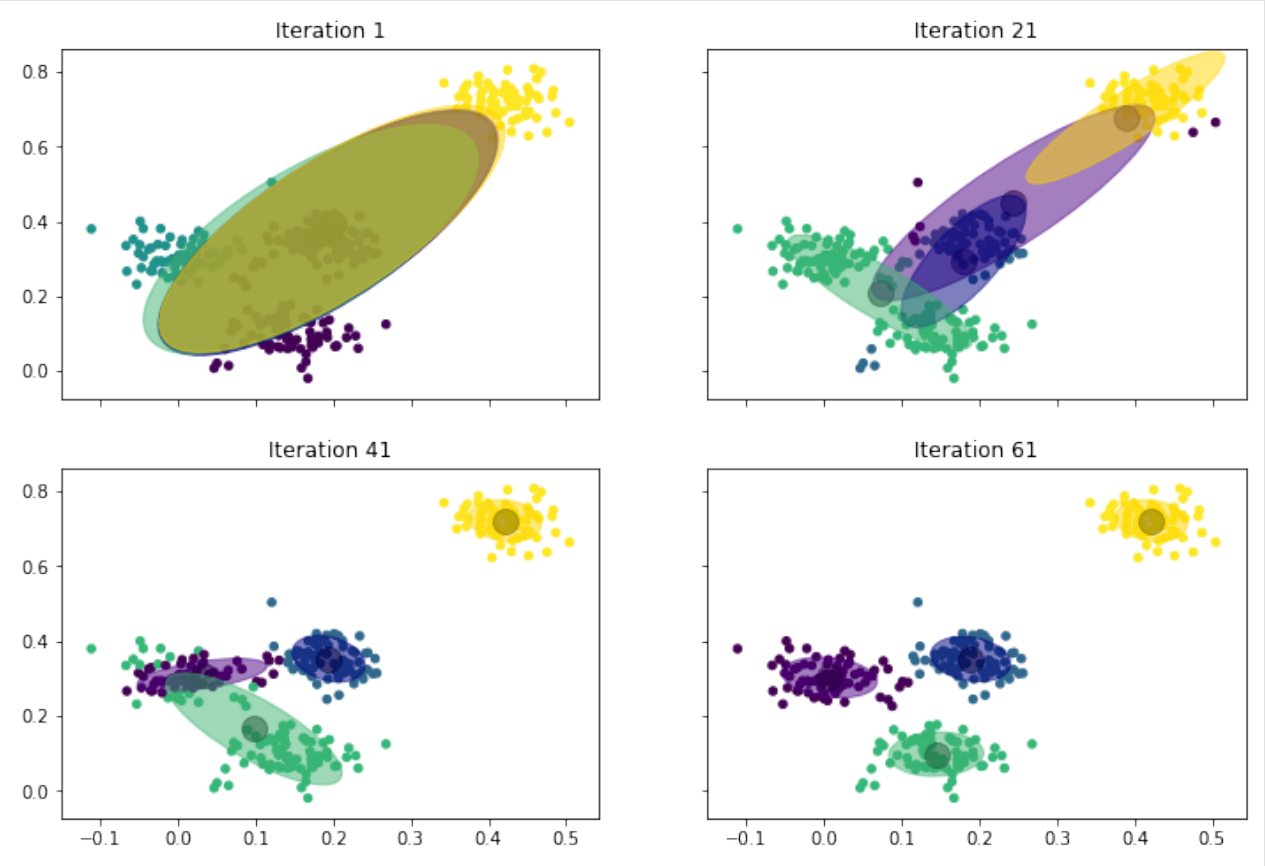
   with $i = 1, ..., k$ as the index of the clusters.

4. Repeat steps 2 and 3 until

- Convergence, i.e., the clusters $C_i, i = 1, ..., k$ do no change anymore.
- A predefined maximum number of iterations is reached.

The biggest difference between the EM algorithm and the k-Means algorithm are the weights. The weights specify the likelihood that an instance $x \in X$ belongs a specific cluster. For example, if $w_i(x) = 0.9$ it means that the likelihood is that x belongs to cluster C_i is 90%. The cluster assignment follows the same rule as above, i.e., x is assigned to the cluster that maximizes the weight $c(x) = max_{i=1,...,k} w_i(x)$. Still, EM also allows insights if points are uncertain, i.e., multiple clusters have similar likelihoods. This is called *soft clustering*, because theoretically each point does not belong to a single cluster, but to all clusters with a certain likelihood.

To better understand how the EM algorithm works, we again look at how the update steps work to fit the data.



This is what happens:

- Iteration 1: We start with four normal distributions that are almost the same. The mean values are in the middle of the data, the covariances are ellipses that are elongated in the diagonal, but slightly different. For example, we can see that the green covariance ellipse goes a bit more to the bottom left, and that the yellow one goes a bit more to the top right. This already shows in the assignment of the instances to the clusters: the instances in the top-right corner belong to the yellow cluster, the instances on the left side belong to the green cluster. The other two clusters are not really visible due to the overlap.
- Iteration 21: After 21 iterations we see that the four normal distributions start to drift apart. The yellow one is pulled into the top-right corner, but still has a large covariance ellipse. The green one actually is "torn" between the instances on the left and the instances at the bottom. The blue one is in the middle with a relatively small covariance. The small covariance is an indicator that the blue cluster will not move strongly anymore and stay in the middle. The purple cluster is also in the middle, but still has a very large covariance, which means that it will likely still move elsewhere.
- Iteration 41: The clusters begin to converge. The yellow cluster finished its journey to the top-right corner and now covers only that region. Similarly, the blue cluster covers the points in the middle. This pushed the purple cluster to the left side, where it is now residing with a considerably smaller covariance than in iteration 21. The green cluster is still torn between the left side and the bottom, but starts to drift to the bottom, because purple now claims most of the left.
- Iteration 61: The clusters converged. Purple now covers the instances on the left and green moved to the instances at the bottom.

In the above example we also see two important properties of EM clustering. First, the convergence is much slower than with k-Means. There are more parameters that must be optimized, i.e., not only the mean values but also the covariances. Additionally, the soft clustering means that instances belong to all clusters. This slows the movement of the mean values, because they get a (weak) pull from many instances, especially when the covariance is large.

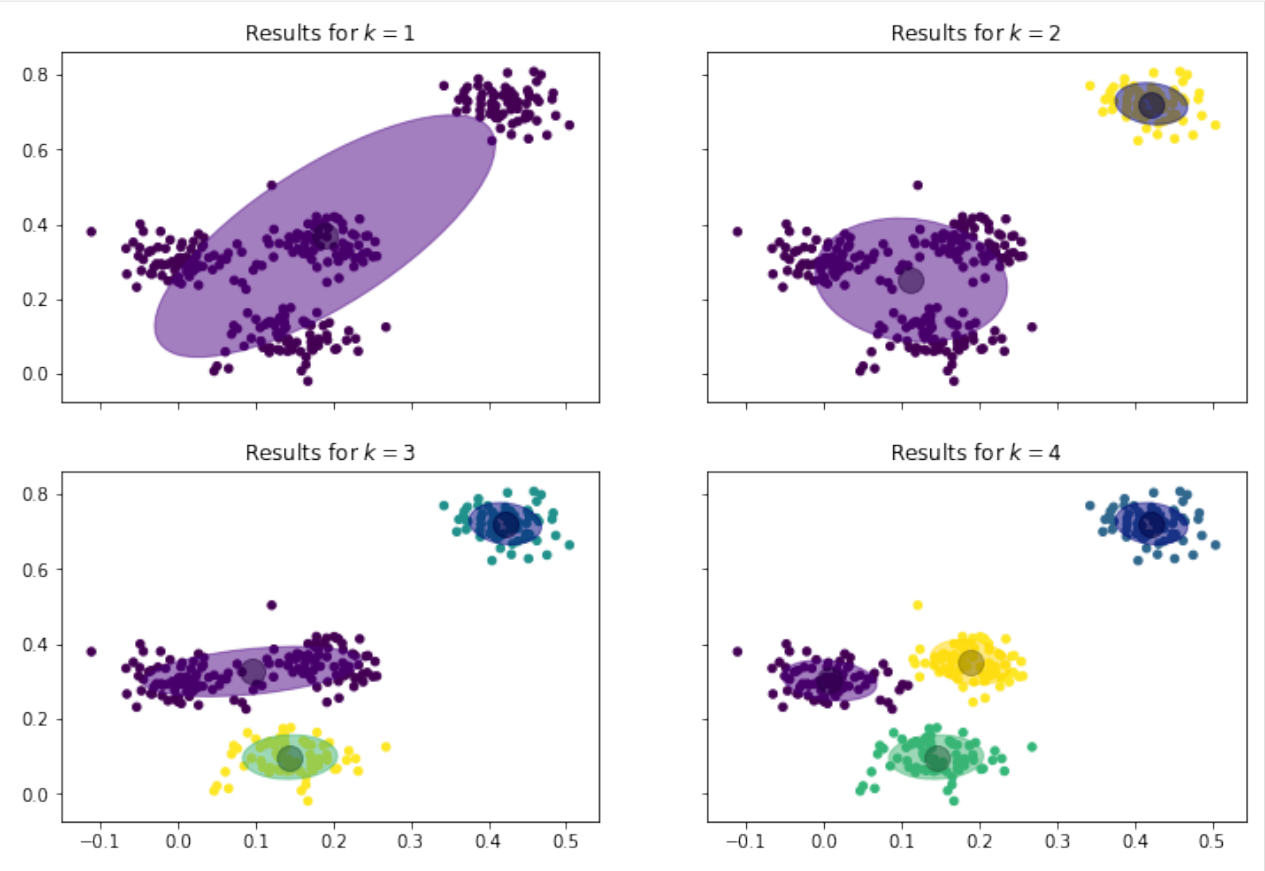
Second, we see that the clusters may cover disjoint regions. For example, in iteration 41, the green cluster contains the instances at the bottom, and some instances on the left side. Between these instances are purple instances, i.e., there are two disjoint regions of the green cluster. This is possible due to the shape of the covariances. Because green has a larger covariance, points that are farther away from the mean have a higher probability of being green than purple, because of the low covariance. This effect of disjoint regions is something that cannot be observed with many clustering algorithms. For example, this is impossible with k-Means, because distances change the same in all directions.

Note:

The similarities between k-Means and EM clustering go beyond the descriptions above. In fact, k-Means is a special case of the EM clustering, in which Voronoi cells are used to model the distribution.

6.3.3 Selecting k

Same as for k-Means, the number of clusters must be determined by the user of the EM clustering. The general approach is the same: domain knowledge and visualizations are important to select appropriate values of k.

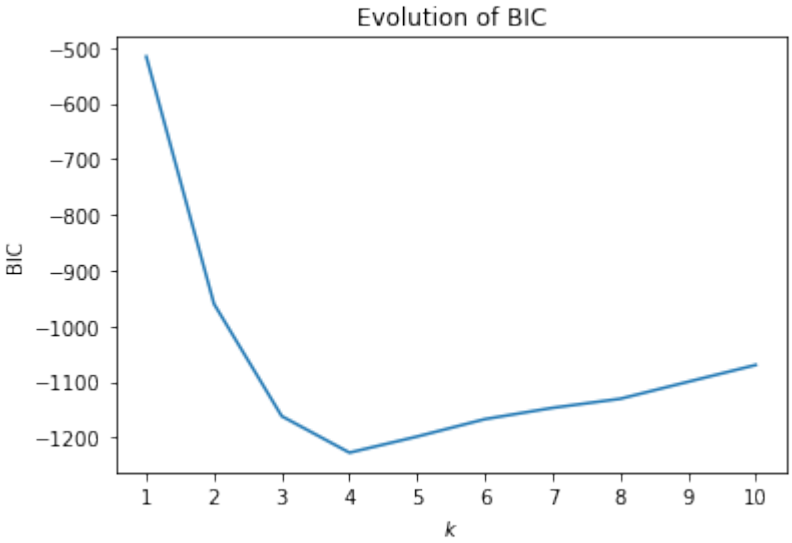


We can see that a simple normal distribution is not sufficient to describe the data, the covariance of the data with only one cluster is huge. The results for $k = 2, 3, 4$ all seem better. However, only with $k = 4$ the clusters seem to have similar covariances.

There is also an analytic approach for the selection k, based on the *Bayesian Information Criterion* (BIC). Similar to the WSS, BIC is a measure for what the algorithm tries to optimize and can be used to determine how different values of k perform with respect to the objective of the clustering algorithm. The EM algorithm maximizes the likelihood, i.e., tries to find distributions such that the data can be explained by the clusters. Formally, this is measured by a likelihood function $\hat{L}(C_1, ..., C_k; X)$ that we have ignored in the discussion of the algorithm so far and that we will also not define in detail here. The likelihood increases, when the clusters are a better explanation for the data. Similar to the WSS, the likelihood usually decreases with more clusters. However, BIC also takes the complexity of the model into account. We mentioned at the beginning, that there are $k' = k \cdot (d + \frac{d \cdot (d+1)}{2})$ parameters if there are d features. Occam's Razor tells us that the simplest explanation is usually correct. This means that if we can achieve a similar likelihood with fewer parameters, we have a simpler explanation and should use this. The more complex information, i.e., more clusters with more parameters, may be overfitting. Based on these ideas, BIC is defined as

$$BIC = \log(|X|) \cdot k' - 2 \cdot \log(\hat{L}(C_1, ..., C_k; X)).$$

Thus, the value increases with more parameters and more training data and decreases if the likelihood increases, i.e., the clusters are a better explanation. Thus, BIC should be minimized. In comparison to WSS, BIC is not monotone but has a minimal value, because BIC increases if the parameters in the model are increasing without an equally strong increase in the likelihood. From this follows that, according to BIC, there is an *optimal* number of clusters that can be automatically determined as the value of k that minimizes BIC. This is demonstrated by the evaluation of BIC for our example.



The minimal value is at $k = 4$ meaning that four clusters are the optimal trade-off between number of parameters required to describe the clusters on the one hand, and the likelihood of the cluster result on the other hand. Please note that the actual values of BIC should usually be ignored. Whether BIC is positive or negative, if the values are in the order of 10^2 or 10^5 usually depends more on the data than on the quality of the clustering result. Consequently, one could even remove the values from the y-axis of the plot.

Note:

BIC is derived from the *Akitake Information Criterion* (AIC), which is defind as

$$AIC = 2 \cdot k' - 2 \cdot \log(\hat{L}(C_1, ..., C_k; X)).$$

AIC is directly derived from the Kullback-Leibler divergence, an information theoretic measure for the difference between random variables. This is also the reason for the logarithms in the AIC formula and, consequently, the BIC formula. The difference between AIC and BIC is only in the first term, where AIC uses a factor of 2 for the number of parameters and BIC uses the logarithm of the size of the data.

6.3.4 Problems of EM Clustering

EM clustering resolves two of the problems of the k-Means algorithm: scale effects and the roundness of clusters. The ranges of scales are irrelevant, because this can be covered by the covariances. Similarly, the EM goes beyond round clusters and can describe ellipses, which are more versatile. Otherwise, EM clustering and k-Means clustering still share problem.

- EM clustering is also sensitive to the initialization. Bad initializations may lead to bad clusters, but also to a very slow convergence speed, i.e., many iterations may be required, which can be computationally expensive and the result may not even converge to a solution. A good approach to initialize the EM algorithm is to actually use the centroids of the k-Means algorithm.
- The choice of k may be bad. However, the BIC yields a good approach for finding a suitable value for k. Please note that sometimes the optimal value for number of clusters can require too many clusters for a reasonable result. In this case, BIC should be interpreted the same way as WSS by looking for elbows.
- Shapes that are not ellipsoid usually cannot be clusters by EM, e.g., the half moons could still not be clustered correctly. In general, EM works best for normally distributed data, and may fail for any other kind of data.

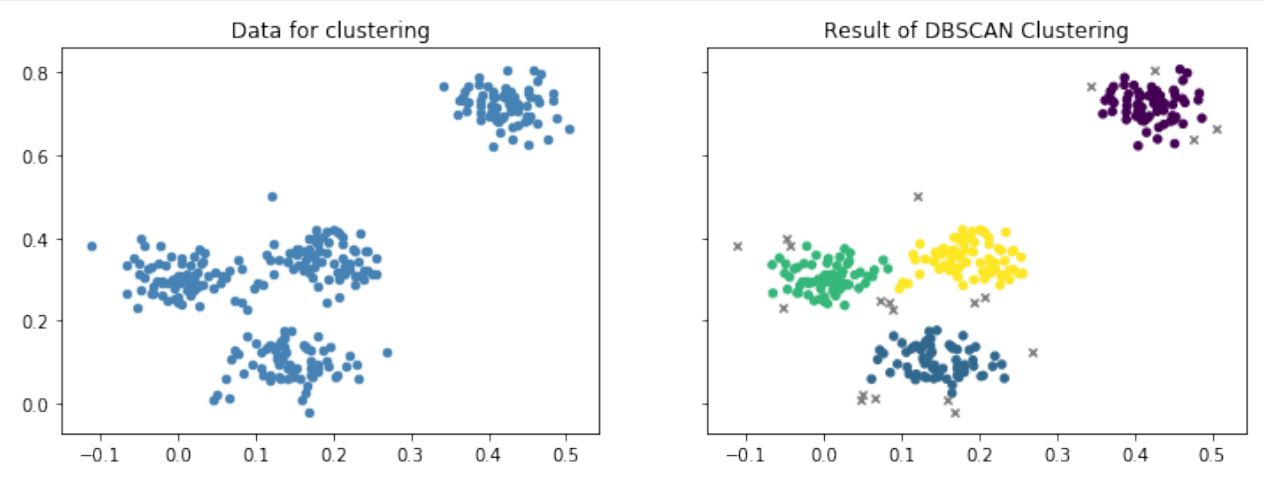
6.4 DBSCAN

6.4.1 General Idea

So far, we relied on the town halls as reference points for the definition of cities. However, we can also ignore the town halls and just look at the neighbors. Just look outside. The houses that are close to your house are your neighbors and belong to the same city. The neighbors of your neighbors as well. And all neighbors of neighbors of .... of neighbors, until all houses that have a direct or indirect neighborhood relationship are part of the city. This is the concept of *density-based clustering*: regions that are *dense*, i.e., where all points are neighbors of each other, as well as all neighbors of neighbors belong to the same cluster.

This approach for clustering is fundamentally different from k-Means and EM clustering, because we do not have a formula that we can use to determine to which cluster an instance belongs. Instead, the instances themselves define the clusters. Thus, if we want to know to which cluster an instance belongs, we must look at the neighbors.

The most common density-based clustering algorithm is DBSCAN. The following figure visualizes the results of the DBSCAN clustering of our sample data.



We can see four clusters that are similar to the results of k-Means and EM clustering. However, there are also some instances that are marked as grey x. These are *noise*, i.e., instance that do not belong to any cluster. This can happen, if instances are not in a dense neighborhood. This is the same as houses that are outside the boundaries of cities.

6.4.2 The Algorithm

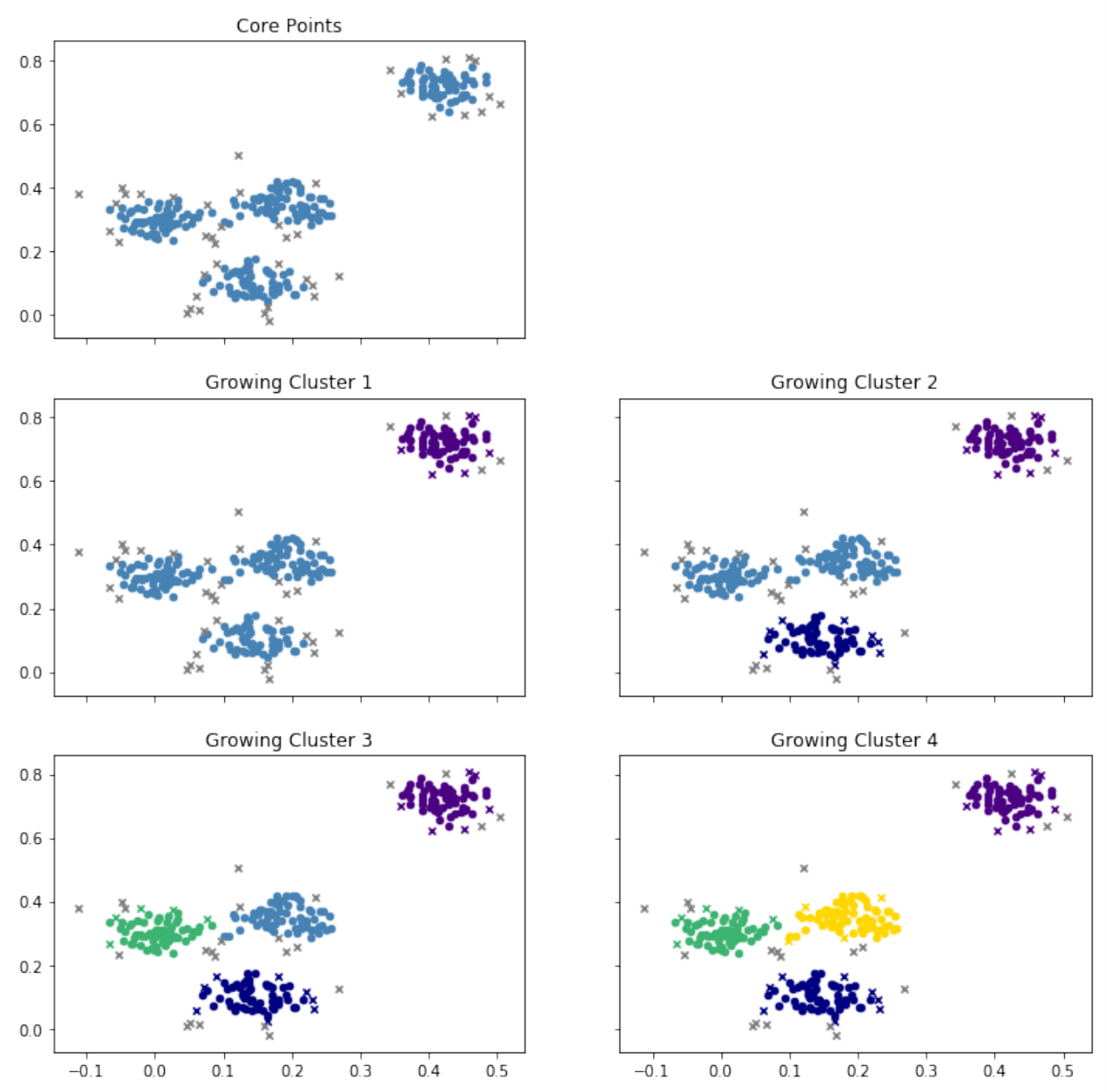
The algorithm for DBSCAN requires a concept for what a *dense neighborhood* is. A neighborhood is defined using a distance $\epsilon \in \mathbb{R}$ such that $neighbors(x) = \{x' \in X : d(x, x') \leq \epsilon\}$. A neighborhood is called dense, if it has more than $minPts \in \mathbb{N}$ points. We call all points $x \in X$ that have a dense neighborhood core points. We use the notation $core(C) = \{x \in X : |neighbors(x)| \geq minPts\}$ to denote all core points within a cluster $C \subset X$, $core(X)$ to denote all core points.

Once we know all core points, we simply *grow* clusters. We randomly pick a core point and define that this core point belongs to the first cluster. We then add all instances that are neighbors of the core point to the cluster. We repeat this for all core points there were added to the cluster, until we have all neighbors of all core points within the cluster. Then, we pick the next core point that was not yet assigned to a cluster and grow the next cluster.

1. Set $i = 1$, i.e., we are working on the first cluster.
2. Pick a core point that does not yet belong to any cluster and initialize a new cluster, i.e. $x \in core(X) \setminus \bigcup_{j_i} C_j$ and initialize $C_i = \{x\}$ to start a new cluster.

3. Add all neighbors of core points in C_i to the cluster, i.e., $C_i = \bigcup_{x \in core(C_i)} neighbors(x)$.
4. Repeat step three until no additional points are added.
5. If there are still core points not yet assigned to a cluster increment $i = i + 1$ and go back to step 2.

The formal explanation of the algorithm is a bit convoluted. How the algorithm works gets clearer when we look at the example.



This is what happens:

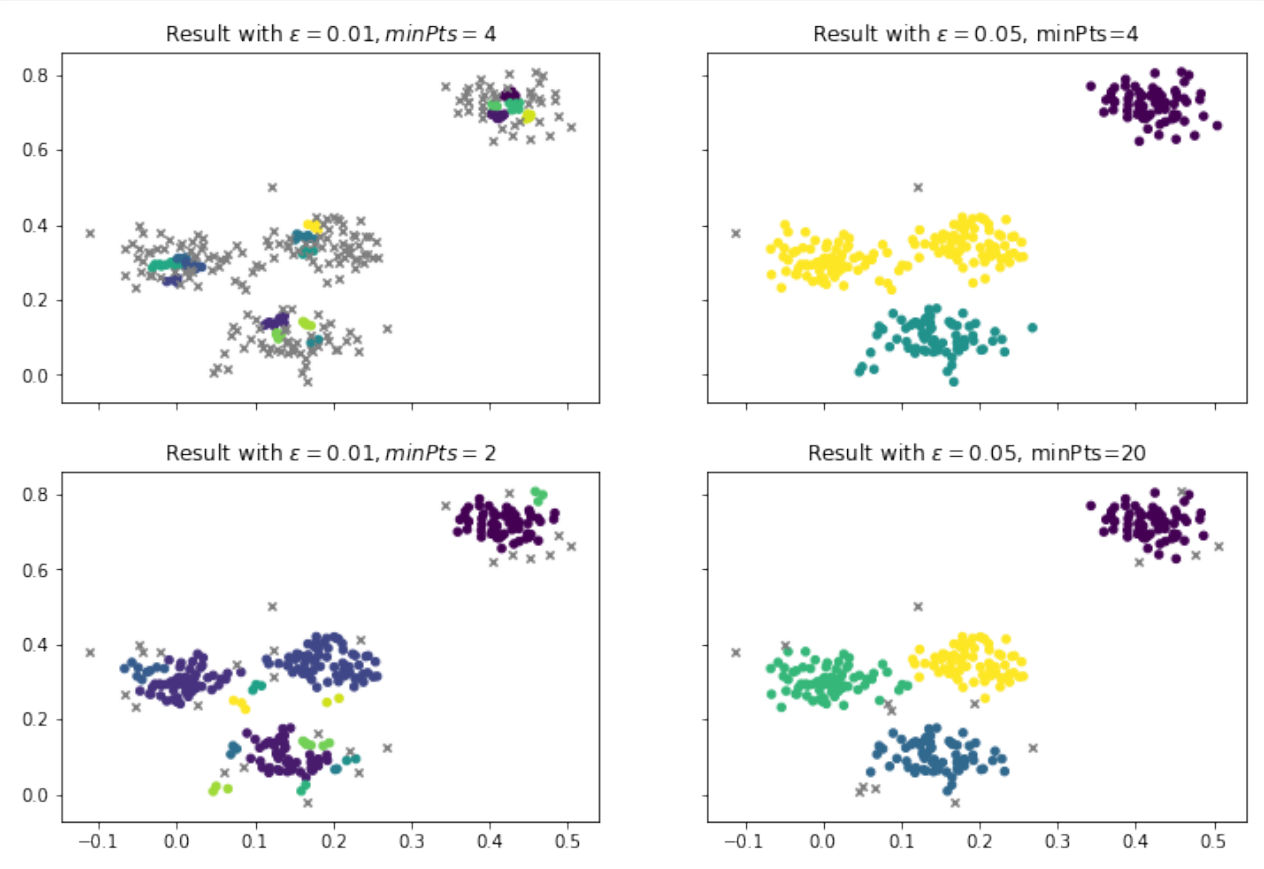
- At the beginning, all core points are determined. These points will definitively be assigned to a cluster later, because they are in a dense neighborhood.
- Then, the first cluster is grown. For this, a core point is picked at random. In the example, the initial core point is one of the core points in the top-right cluster. Then all other core points in the region are added, but also some points that are not core points, marked as purple x. These points are in the neighborhood of a core point, but not

core points themselves. You may think of these points at the houses at the boundary of a city. There is also some grey x left in the top-right area. These are not in the neighborhood of any core point and will become noise.

- A new core point is picked randomly to grow a second cluster, this time at the bottom. We see the same thing happening again. All core points in the region, as well as some non-core points are added, marked as blue x. Others are not close enough to any core point and left as noise.
- This is repeated twice more, first with the green cluster on the left and, finally, with the yellow cluster in the middle.
- Since all core points are now assigned to one of the four clusters, the DBSCAN algorithm stops. All instances that are still marked with a grey x are noise.

6.4.3 Picking ϵ and $minPts$

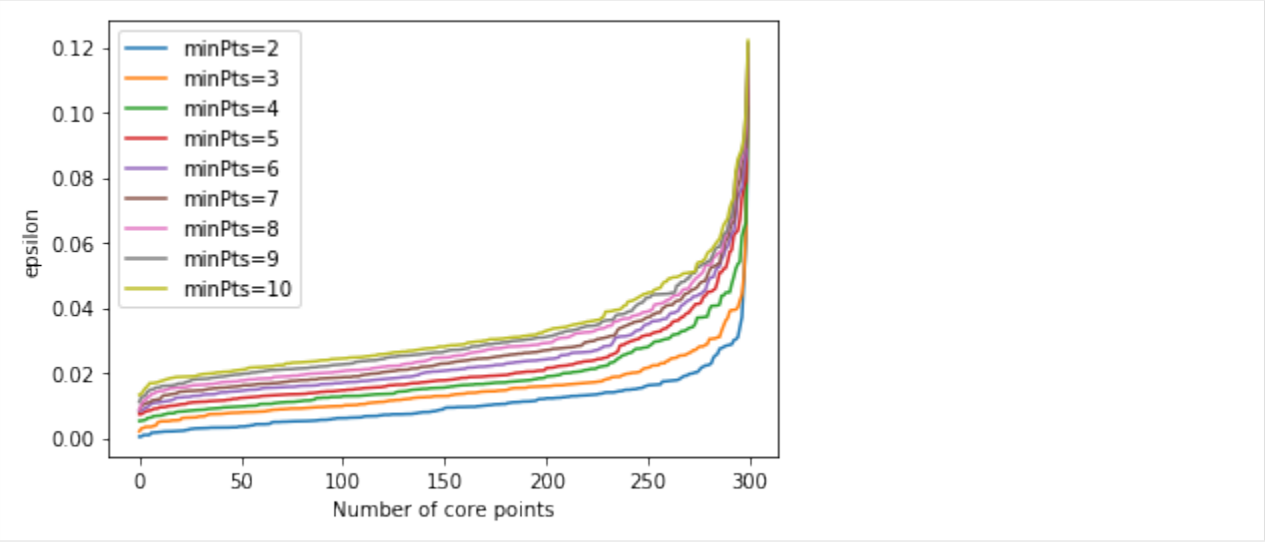
An advantage of DBSCAN is that the number of clusters does not have to be defined. Instead, this is automatically determined based on the neighborhoods of the core points. Instead, the two parameters ϵ and $minPts$ are required, to define what a neighbor is and how many neighbors are required for density. Unsuitable values may lead to bad results. For our example, we used $\epsilon = 0.03$ and $minPts = 4$. Below are some results that are achieved with other values.



In the first column, we have a very low ϵ of 0.01. If we keep the minPts at 4, we find no real clusters, i.e., there are almost no core points anymore. When we decrease the minPts to 2, we find clusters again. There are four larger clusters that look similar to our initial result. However, there are also many smaller clusters with only three points. Thus, this is not a good result, because such small clusters usually do not make sense and should either be merged into larger clusters or considered as noise. The second column shows what happens when we use a large ϵ of 0.05. The results actually look good and similar to our prior results. With minPts at 4, we find the three clusters we can also find with $k = 3$ with k-Means and EM clustering. When we increase the minPts to 20, we have fewer core points

that have larger neighborhoods. This leads to almost the same result as we have with $\epsilon = 0.03$ and $minPts = 4$, but with less noise. In general, reducing ϵ will always lead to more and smaller clusters, increasing ϵ will merge existing clusters. Reducing minPts usually leads to more clusters, but may also allow very small clusters that do not make sense. Additionally, reducing minPts may also merge clusters that are close to each other, because there are more core points that could serve as a bridge between the clusters. Increasing minPts avoids small clusters but can also lead to no clusters or lots of noise if ϵ is too large. Thus, both parameters should always be used together and there is no general rule for their optimization.

However, there is one analytic tool other than visualizing the clusters that can help to determine good values for ϵ. We can plot how far away the k-th nearest neighbor of instances are.

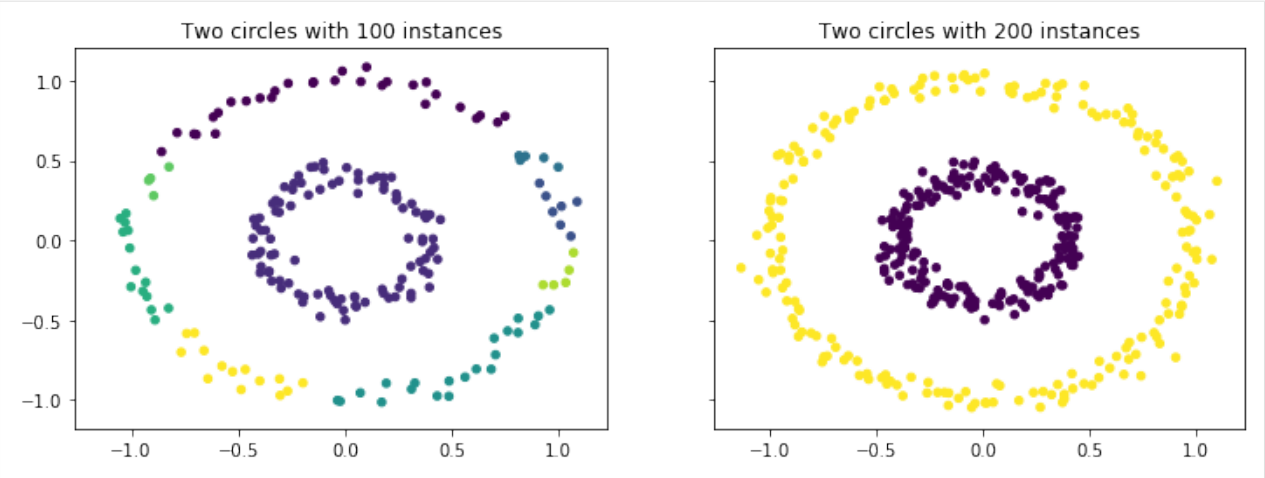


Such a plot shows us, how many core points we would have for a combination min. A good value for ϵ is where the curvature is largest, i.e., where you see that sharpest change of direction in the curve. For example, we see that this is somewhere in the between 0.3 and 0.4 with $minPts = 4$.

6.4.4 Problems of DBSCAN

While DBSCAN does not have the problem that the number of clusters must be defined by the user and can also find clusters of any shape, there are other aspects of the algorithm that may be problematic.

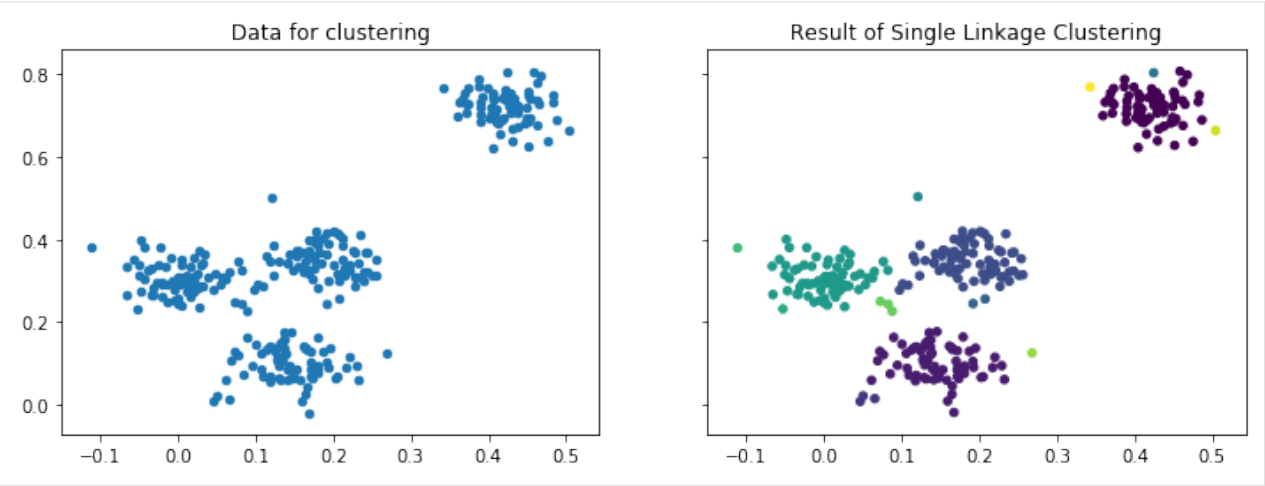
- The biggest problem of DBSCAN is that it can be difficult to find good values for ϵ and $minPts$. Especially if clusters are very close to each other, it can be very difficult to separate the data into two clusters. The reason is that the instances where the clusters are close to each other act as *bridge points* between the clusters. This can only be avoided by using a large value for minPts that decreases the likelihood that points at the border of clusters are core points. However, this means that ϵ must be relatively high, which can also lead to problems.
- Another problem is that clusters may have different densities. The example below shows the clustering of circles. With only 100 instances per circle, the data in the outer circle is not very dense in comparison to the inner circle. As a consequence, there are some breaks in the outer circle and it is not detected as a single cluster, but rather as many clusters. When the instances in the outer circle are increased, both clusters are detected correctly. This problem can be quite severe, if every cluster has different densities.



- Consequently, you cannot easily apply DBSCAN to a subsample of data, because the values for ϵ and minPts depend on the density of the sample which usually decreases when a subsample is used.
- DBSCAN is also distance based and as such sensitive to scale effects in the same way as k-Means clustering.

6.5 Single Linkage Clustering

There is another way how cities often grow, i.e., by merging with other cities and villages into a larger city. This is always done by cities that are close to each other. This is the concept of *hierarchical clustering*. The idea is to build clusters by growing them, similar to DBSCAN, but not by considering dense neighborhoods, but instead by merging of existing clusters into larger clusters. The outcome of hierarchical cluster are many different possible cluster results, because the merging of clusters can continue until all instances are in a single cluster. The result of hierarchical clustering may look as follows.



The result looks similar to the outcome of the DBSCAN clustering, but there is one key difference. There is no noise, instead there are some very small clusters, some even with only a single instance, e.g., the left-most instance.

The difference between hierarchical clustering algorithms is how the clusters are determined by merging data. We note that there are also hierarchical clustering algorithms that work the other way around, i.e., by splitting larger clusters into smaller clusters, e.g., *Complete Linkage Clustering*.

However, in the following, we focus on the *Single Linkage Clustering* (SLINK) algorithm that creates clusters through merging smaller clusters.

6.5.1 The SLINK Algorithm

Single Linkage Clustering (SLINK) iteratively merges the two clusters that are closest to each other and remembers the clusters and their current level. SLINK starts with every instance in a cluster of their own. Then, clusters are merged based on their distance to each other. The distance of two clusters C and C' is defined as the distance between the two instances in the clusters that are closest to each other, i.e.,

$$d(C, C') = \min_{x \in C, x' \in X'} dist(x, x').$$

Each cluster that is determined by SLINK has a *level*. The level is defined as the distance of the two clusters that are merged. Thus, the level can be seen as the distance between the two cities that are merged. Consequently, a low level means that the clusters that are combined are not far away from each other, a higher level indicates that there is some gap between the clusters that were merged.

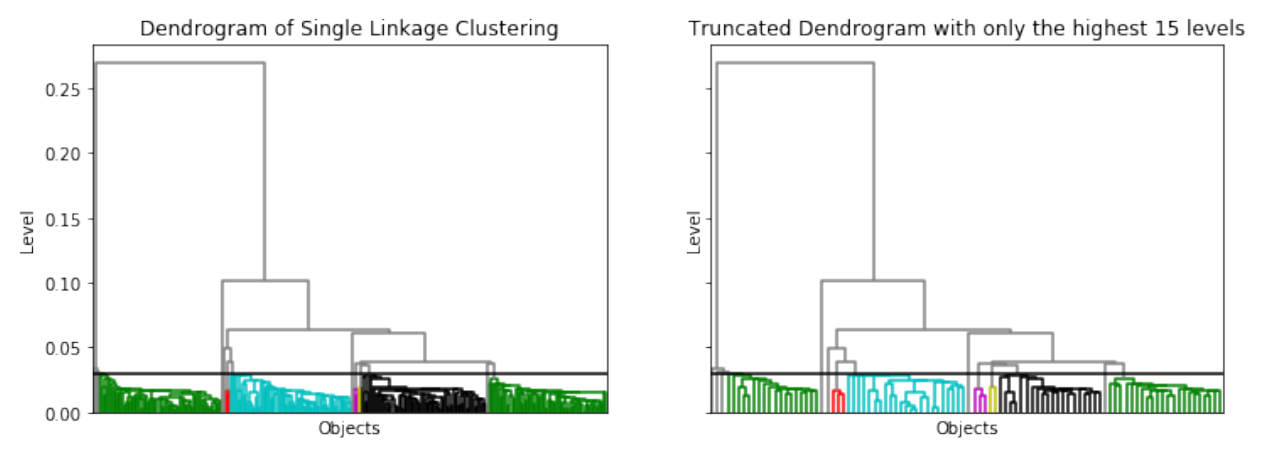
Based on these ideas, we get a relatively simple algorithm.

1. Initialize basic clusters $C = \{x\}$ for all $x \in X$ with level $L(C) = 0$.
2. Find the two clusters that are closest to each, i.e., $C, C' = \arg\min_{C,C'} d(C, C')$
3. Merge C and C' into a new cluster $C_{new} = C \cup C'$ with level $L(C_{new}) = d(C, C')$.
4. Repeat steps 2. and 3. until all instances are in one cluster.

The different clusters and how the result can be interpreted is best evaluated through visualization with a *dendrogram*.

6.5.2 Dendrograms

A dendrogram is, in general, the visualization of a tree data structure. For our example, the dendrogram looks as follows.

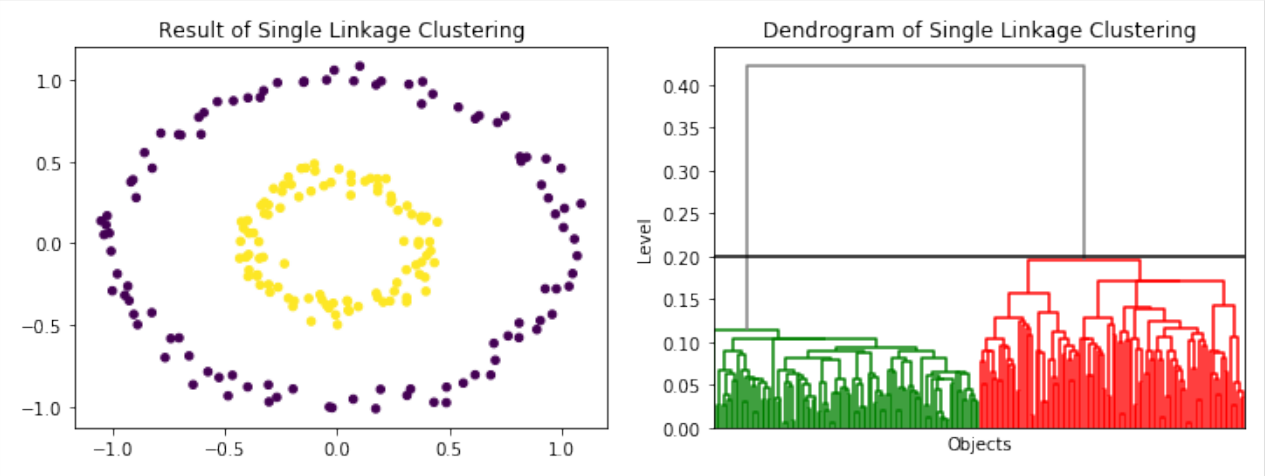


At the bottom of the dendrogram, we have all the instances, i.e., the clusters of level 0. Each horizontal line connects two clusters into a new cluster. The location of the horizontal lines on the y-axis is the level of the new cluster. For example, the horizontal line at 0.1 combines all instances on the left side of the tree with all instances on the right side of the tree into a new cluster of level 0.1. From this, we also know that the minimal distance of the instance on the left side of that tree to all instances on the right side of that tree is 0.1. The black horizontal line is at 0.03. The colors below indicate the resulting clusters, i.e., the clusters with a level of at most 0.03. The left figure is very dense at the bottom and we basically see large colored regions. This is because we have all instances at the bottom, in this case 300. A trick to make the bottom of the tree "readable" is to drop the *lower layers* of the tree. This means that we do not visualize clusters of all levels, but only the highest levels of clusters. For example, the right figure shows only the 15 highest levels of clusters. This way, we do not see the individual instances at the bottom of the dendrogram, but rather relatively small clusters. As a result, we can see the structure of the dendrogram.

Working with dendrograms takes some getting used to. However, they provide a great tool to visually determine clusters manually. In the above example, we used a fixed threshold of 0.03 for the merging of clusters. We also see that the black and the green cluster on the right would merge, if we would change this threshold to 0.05. If we choose a cutoff between 0.06 and 0.25, we would get two large clusters. For a cutoff betwee 0.06 and 0.1, there would be still be a third cluster, that would only have a single instance. This is a clear advantage of SLINK over DBSCAN, where picking good values for ϵ and minPts can be quite difficult.

6.5.3 Problems of SLINK

- The biggest problem of SLINK (and hierarchical clustering in general) is the scalability for large data sets. The standard algorithms require a square matrix of the distances between all instances. For 100,000 instances, this matrix already requires 4 Gigabyte of storage, even if we only store the upper triangle of the matrix. Thus, the application of hierarchical clustering is usually limited to smaller data sets and this approach is not suitable for big data.
- Hierarchical clustering has the same problems as DBSCAN with respect to different densities. However, it is easier to find a good cutoff, that separates the different clusters. For example, for the circles we see directly from the dendrogram that any cutoff between 0.2 and 0.4 works to find the correct clusters.



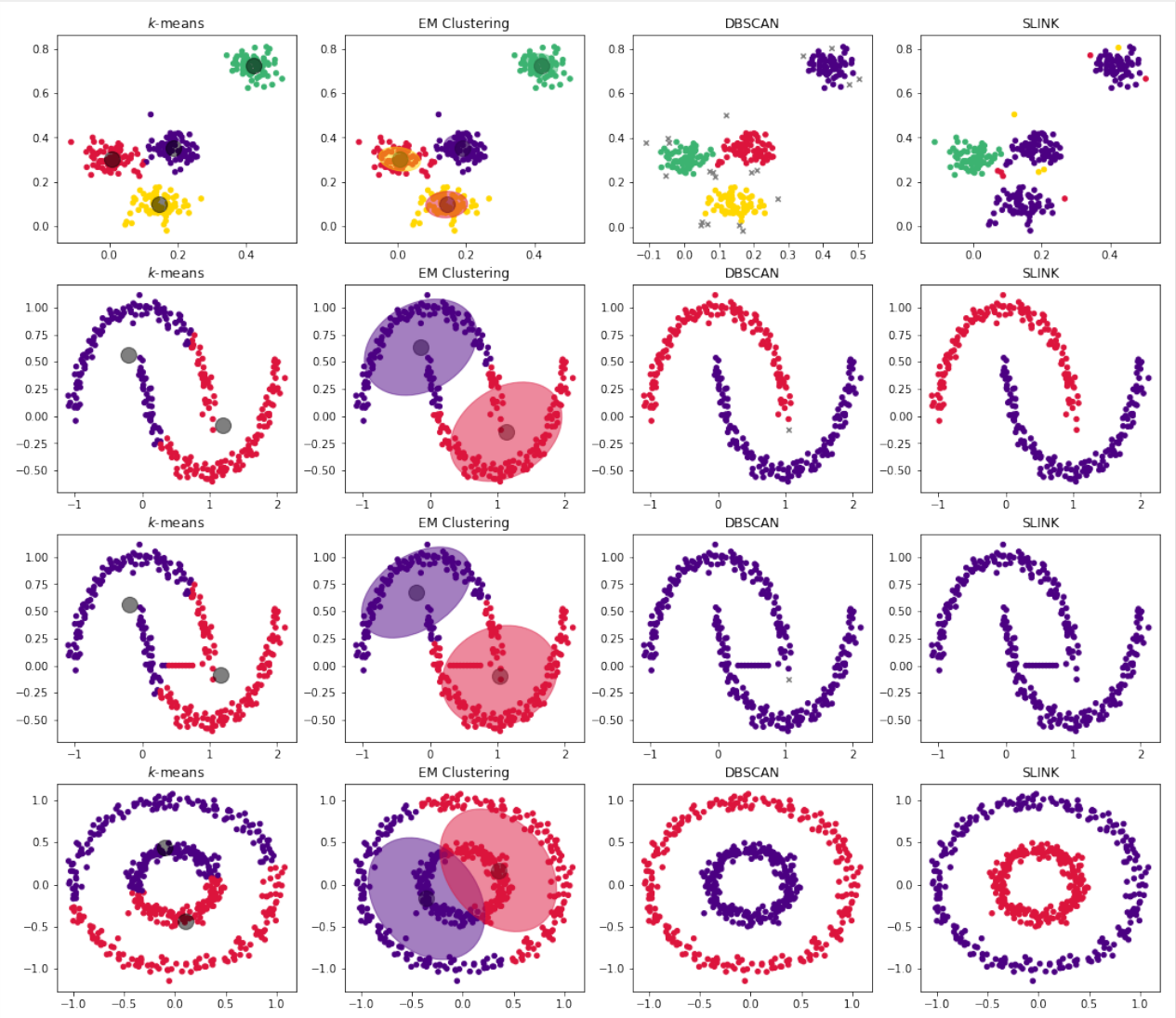
- Bridge points between clusters can also have a stronger effect on SLINK than on DBSCAN. The reason is that DBSCAN and SLINK yield the same clusters, if minPts is set to 1. Then, the threshold for the level becomes the same as ϵ. However, this means that using a higher value of minPts to mitigate problems with the merging of clusters due to bridge points between two clusters points cannot be used with SLINK clustering.
- SLINK may lead to many very small clusters, because all instances start in a cluster on their own. If they are outliers, they will still get a cluster, which may make the results hard to interpret. DBSCAN uses the minPts and the noise to mitigate this.
- SLINK is also distance based and as such sensitive to scale effects in the same way as k-Means clustering.

6.6 Comparison of the Clustering Algorithms

We already partially compared the clustering algorithms above, especially how some algorithms work around the problems of other algorithms, but also which problems they share. In the final part of this chapter, we compare the algorithms directly.

6.6.1 Cluster Shapes

First, we consider the outcome of the clustering with good parameters on data sets with clusters of different shapes. Below, we see the results for our running example, as well as the circles and half-moons we used to show problems before.



We can see that on relatively simple data, like our running example, all algorithms yield good results and the choice of the clustering algorithm does not matter much. However, there are small differences at the boundaries of the clusters, e.g., the seemingly random borders between green and purple for k-Means and EM clustering, the noise for DBSCAN, and the small clusters for SLINK.

The second row shows the results for the half-moons. We see what we already discussed above: k-Means can only

find more or less circular clusters and EM ellipsoid clusters. Thus, both fail on that data and the clusters do not make sense. DBSCAN and SLINK work well, because they build their clusters on pair-wise neighborship of instances, which works with any shape of clusters, as long there is a gap between the clusters. This can be broken by a "bridge" between the two moons is demonstrated in the third row.

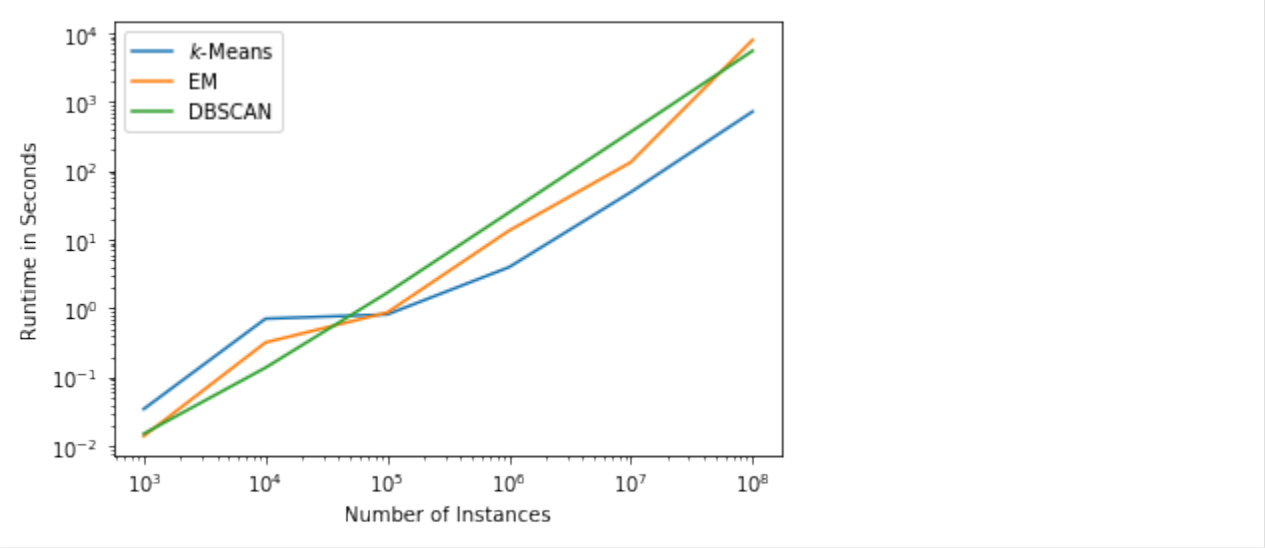
The fourth row demonstrates that while k-Means and EM clustering work in general for round shapes, it is also important that these round shapes are not within each other. If we have a circle within a circle, this is also not correctly identified by the clustering.

6.6.2 Number of Clusters

Another strength of DBSCAN and SLINK clustering is that they can determine the number of clusters automatically, just by knowing the definition of core points, respectively the maximum level. EM clustering also provides good support to determine the number of clusters with the BIC, which has a minimal value that is optimal. The WSS may be used for k-Means, but looking for elbows in a plot is very fuzzy and requires a high expertise by the data scientists.

6.6.3 Execution Time

While the cluster results are certainly important, the execution time can also be a factor. Below are the results of the execution time of clustering data with 20 features for 10,000, 100,000, 1,000,000, 10,000,000, and 100,000,000 instances, i.e. up to about 15 Gigabytes of data. Please note that we omit SLINK from this comparison, because it already crashes with 100,000 instances. The measurements are done with a normal laptop (Intel Core i7-8850 @ 2.60GHz, 32 GB RAM) with scikit-learn 0.23.1.



We can see that the runtime of all three algorithms is comparable, even though DBSCAN is a bit slower. We note that this also depends on the data set. Sparse data, dense data, or strange distributions may alter these estimations. However, in general runtime is not a major concern for the selection of one of these clustering algorithms.

6.6.4 Explanatory Value and Concise Representation

A key advantage of k-Means clustering and EM clustering is that they provide a concise mathematical representation that describes each cluster, i.e., the centers and the normal distributions, respectively. This is a great tool for data scientist for the interpretation of the meaning of clusters. Moreover, the concise representation can be be implemented elsewhere to assign instances to clusters, without the need to apply the clustering again.

DBSCAN and SLINK both do not provide a mathematical representation of the clusters. Instead, the clusters are only described by the instances themselves. This also hinders the interpretation, which can almost only be done visually. SLINK supports this better, because of the dendrograms which give an overview of the distances between the clusters.

6.6.5 Categorical Features

Categorical features are a general problem for all of the clustering algorithms discussed above. For k-Means, DBSCAN, and SLINK the reason is that they are based on distance measures. However, by definition, the distance between different categories is unknown. Thus, we cannot compute reasonable distances. While there is no good solution for DBSCAN and SLINK, there is the option to use k-Modes clustering, which is the same as k-Means, but with the mode instead of the arithmetic mean.

The reason for EM clustering with normal distributions is similar. There is no meaningful way to describe categorical data with a continuous distribution. The solution is to use EM clustering with discrete distributions, e.g., a multinomial distribution.

6.6.6 Missing Features

None of the algorithms works if there are missing feature values, i.e., an incomplete description of an instance. Thus, either the missing instance or the missing feature must be discarded. Alternatively, feature imputation may be used to guess the value of the feature (e.g., Barnard and Meng, 1999).

6.6.7 Correlated Features

Correlated features are also problematic for all algorithms, but especially for the distance-based approaches. An extreme example would be if you have twice almost the same feature, e.g., the normalized age in years and the normalized age in days. Normalized means that the values of the feature are rescaled to be in the interval $[0, 1]$. The values of both features will not be identical, because years are coarser than days, but they will be very similar and effectively we only have one feature age. In the distance measure the age would be used twice and, consequently, have disproportional influence on the distance. The more correlations there are in the data, the stronger the potential negative effect gets.

EM clustering can, in principle, deal with the correlations, because they can be represented by the covariances. However, there is still the problem that the clusters get unnecessarily complex, which may skew the BIC and lead to a wrong estimation of a good number of clusters. The reason for this is that the covariance matrix grows quadratically and the model complexity is needlessly penalized by BIC if there are correlated features.

6.6.8 Summary

The following two tables summarizes the different strengths and weaknesses of the clustering algorithms.

	Cluster Shape	Cluster Number	Runtime	Explanatory Value	Concise Representation
k-Means	−	−	o	+	+
EM Clustering	o	o	o	+	+
DBSCAN	+	+	o	−	−
SLINK	+	o	o	+	−

	Categorical Features	Missing Features	Correlated Features
k-Means	−	−	−
EM Clustering	−	−	o
DBSCAN	−	−	−
SLINK	−	−	−

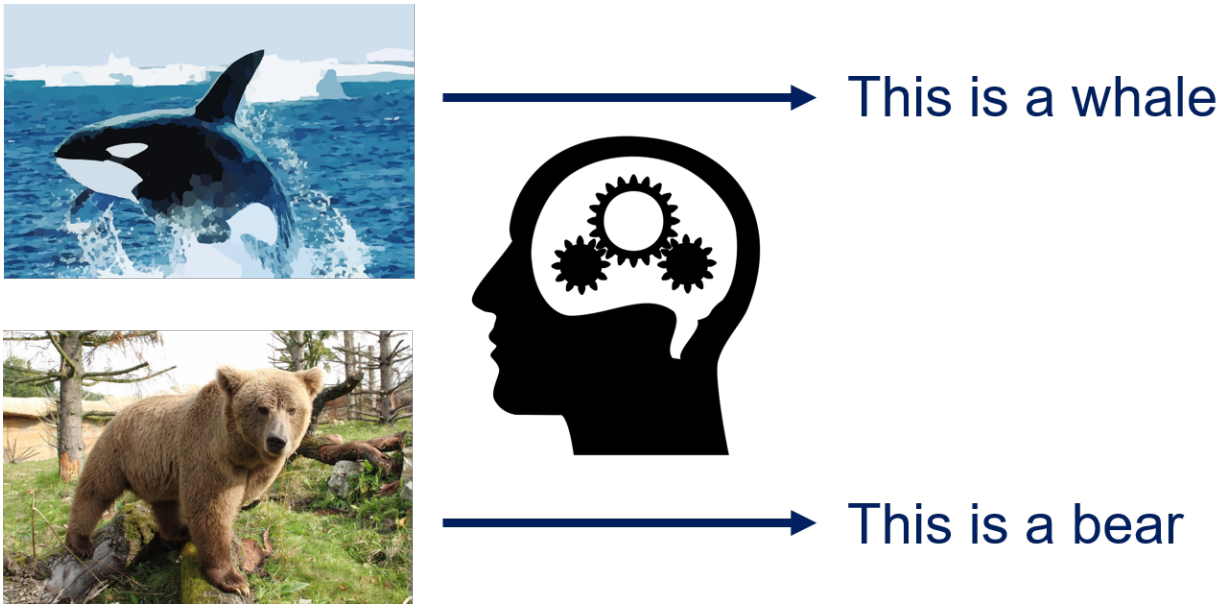
CHAPTER

SEVEN

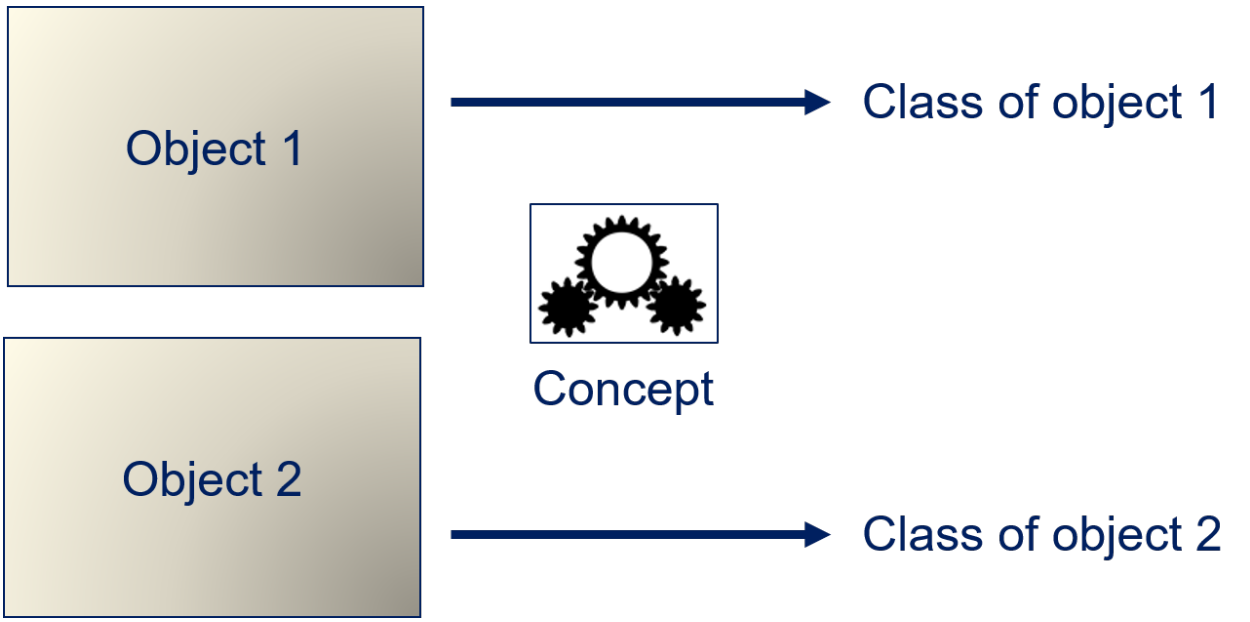
CLASSIFICATION

7.1 Overview

Classification is about assigning categories to objects. Consider the following example.



We have two images, one shows a whale in the sea in front of an iceberg, the other shows a bear in the woods. When we look at these pictures, we immediately see this, because we instinctively assign these categories to the images. Classification is a less powerful approximation of what we instinctively do. The main difference is that we need to work with a fixed and known set of categories. For example, our categories could be "whale picture", "bear picture", and "other image". The images can then be assigned to one of these three categories. The categories to which classification algorithms assign objects are commonly referred to as *classes*. Categories that are not part of the classes, e.g., if there is water in the image, are ignored. A bit more abstract, classification can be described as follows.



We have objects for which we know a *concept*. When we apply our concept to the objects, we get the categories. For example, we have a concept that describes whales, which we can apply to images to determine if something is within the category whale. The task of classification algorithms is to derive a hypothesis that we can use to infer the class of objects from their features. Let us consider the features of the whale pictures.

Based on these features, we may derive the following hypothesis: *"Objects with fins, an oval general shape that are black on top and white on the bottom in front of a blue background are whales."* This hypothesis works reasonably well, even though there may also be other objects that fit the description, e.g., a submarine with a black/white painting. This general approach is used by all classification algorithms. The form of the hypothesis, as well as the way the hypothesis is derived from data depends on the algorithm.

7.1.1 The Formal Problem

Formally, we have a set of objects $O = \{object_1, object_2, ...\}$ that may be infinite. Moreover, we have representations of these objects in a feature space $\mathcal{F} = \{\phi(o) : o \in O\}$ and a finite set of classes $C = \{class_1, ..., class_n\}$.

The classification is defined by a *target concept* that maps objects to classes, i.e.,

$$h^* : O \to C.$$

The target concept is our ground truth, i.e., a perfect assignment of objects to the classes. Usually, we have no mathematical description for the target concept. For example, there is no such mathematical description for the classification of images as whale pictures and bear pictures. The *hypothesis* maps features to classes

$$h : \mathcal{F} \to C.$$

The hypothesis is determined by a classification algorithm with the goal to approximate the target concept such that

$$h^*(o) \approx h(\phi(o)).$$

7.1.2 Scores

A variant of classification is that the hypothesis computes *scores* for each class $c \in C$. In this case, we have a scoring function for each class of the form

$$h'_c : \mathcal{F} \to \mathbb{R}.$$

Scores are similar to *soft clustering*: instead of deciding for only a single class, the classification determines a value for each class, which we can use to evaluate how certain the algorithm is with the decision for a class. When we want to assign the class based on the scores, we usually just assign the class with the highest score. Thus, we have

$$h(x) = \arg\max_{c \in C} h'_c(x)$$

for $x \in \mathcal{F}$.

Often, the scores are probability distributions, i.e., the scores for each class are in the interval [0,1] and the sum of all scores is 1, i.e.,

$$\sum_{c \in C} h'_c(x) = 1$$

for all $x \in \mathcal{F}$.

7.1.3 Binary Classification and Thresholds

A special case of classification problems is where we have exactly two classes. While this is a strong restriction, there are many problems that can be solved using binary classification. For example, the prediction if a borrower will pay back money, the prediction if a transaction is fraudulent, or the prediction of whether an email is spam or not.

For binary classification, we usually say that one class is *positive* and the other class is *negative*. Thus, we have exactly two classes. If we have only two classes $C = \{positive, negative\}$ and the scores, we can calculate the score of one class based on the score of the other class, in case the scores are a probability distribution, i.e.,

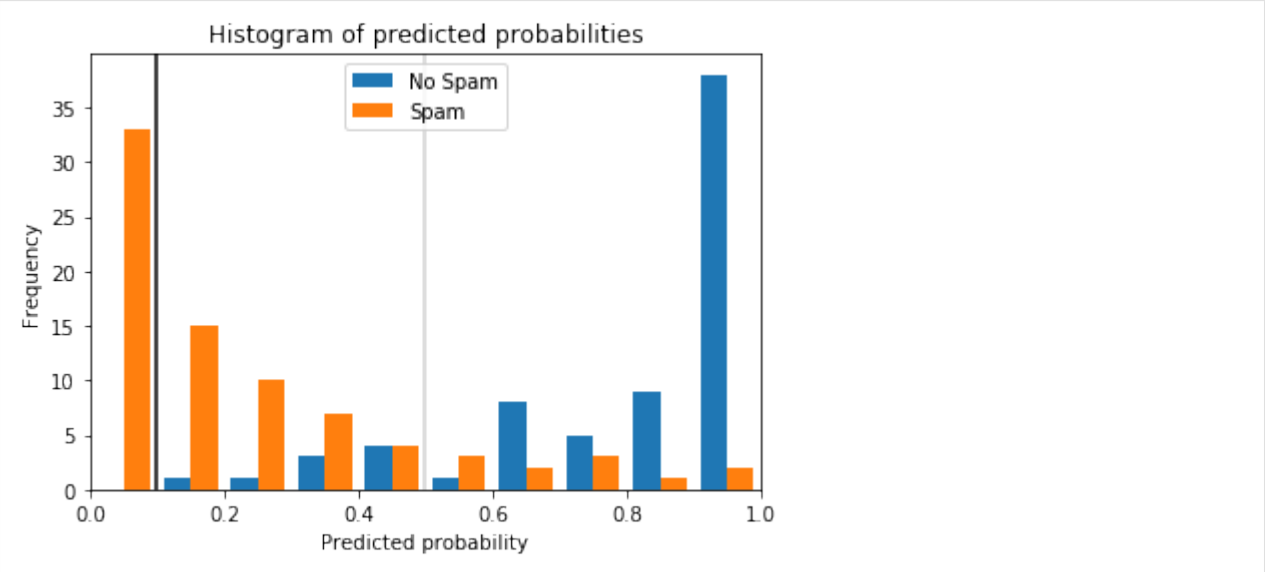
$$h'_{negative}(x) = 1 - h'_{positive}(x)$$

because the sum of the probabilities is one. Because it is sufficient to use the scoring function for the $positive$, we use the notation $h'(x) = h_{positive}$ for binary classification. In this case, we can also use a *threshold* $t \in [0, 1]$ to determine

the classes from the scoring function instead of just taking the class with the highest score. If $h'(x) \geq t$, x is positive, if the score is less than the threshold it is negative, i.e.,

$$h_t(x) = \begin{cases} \text{positive} & \text{if } h'(x) \geq t \\ \text{negative} & \text{if } h' < t \end{cases}$$

Why thresholds and scoring functions are important for classification is best demonstrated through an example. The histogram below shows the scores of instances of a spam detection simulation where positive means that an email is spam.



The orange bars show the scores of spam emails, the blue bars the scores of other emails. Without picking a specific threshold, we would just predict the class with the highest score. This is the same as a threshold of 0.5, indicated by the gray line. This would mean that most emails would be predicted correctly but there would be some emails that would not be flagged as spam, even though they are, and some emails that are flagged as spam, even though they are not. These are different types of errors, and they are not equal in this use case. While spam is annoying, deletion of the spam emails is not a lot of effort, unless there are hundreds of spam emails. On the other hand, even a single email that is mistakenly flagged as spam and not shown to the recipient can have strong negative consequences. We can solve this problem by picking a suitable threshold. The black line indicates a threshold of 0.1. With this threshold, only spam emails would be flagged as spam. While more spam emails would not be detected, at least all normal emails would pass through the spam filter. Thus, classification with scoring and a suitable threshold can make the difference between solving a problem and building a model that is unsuitable for the use case.

7.2 Performance Metrics

The key question of classification is how good the hypothesis h approximates the target concept h^*. Usually, we do not get a perfect solution, which means that there are some instances that are predicted incorrectly by the hypothesis. The spam example above already demonstrates that there are different kinds of mistakes.

The basis for any performance evaluation of classification models is the use of test data. The hypothesis is applied to the features of the test data. From this, we get a prediction result we can compare with actual classes. The following table shows the five features of two instances of our image classification example, the actual class, and the prediction.

shape	top color	bottom color	background color	has fins	class	prediction
oval	black	black	blue	true	whale	whale
rectangle	brown	brown	green	false	bear	whale
...	...	...	...	...	...	...

The first instance is predicted correctly, the prediction of the second instance is incorrect. If there are thousands or even millions of instances in the test data, we cannot evaluate the prediction by looking at such a table. Instead, we need to summarize the comparison between the classes and the prediction.

7.2.1 The Confusion Matrix

The most important tool for the analysis of the quality of hypothesis is the *confusion matrix*, a tabular description of how often the hypothesis is correct and how often it is incorrect, i.e., confused. The confusion matrix for our image classification example may look like this.

		actual class		
		whale	bear	other
Predicted class	whale	29	1	3
	bear	2	22	13
	other	4	11	51

The confusion matrix basically counts how often each instance of each class is predicted as which class. For example, how often whales are predicted as whales, how often they are predicted as bears, and how often they are predicted as something else. The columns are the actual values of the classes, i.e., the target concept. The rows are the predicted values, i.e., the hypothesis. In the example, we have 35 actual pictures of whales. This is the sum of the values in the first column. 29 of these whale pictures are predicted correctly, 2 are incorrectly predicted as bears, 4 are incorrectly predicted as something else. Thus, the confusion matrix gives us detailed statistical information about how many instances we have and how they are predicted. Values on the diagonal of the confusion matrix are the correct prediction, the other values show incorrect predictions.

7.2.2 The Binary Confusion Matrix

The binary confusion matrix is a special case of the confusion matrix for binary classification problems with the classes true and false. In general, the binary confusion matrix looks like this.

		actual class	
		true	false
Predicted class	true	true positive (TP)	false positive (FP)
	false	false negative (FN)	true negative (FN)

Thus, we have actually positive and negative classes and depending on whether the prediction is correct or not, we get true positives (TP), true negatives (TN), false positives (FP), or false negatives (FN). The binary confusion matrix is well known and not only used for the evaluation of machine learning, but also, e.g., in medical studies to evaluate the quality of tests. From medical studies also originate the terms *type I error* and *type II Error*. The type I error measures the false positives. In medicine, this could mean a mistakenly positive result of an antibody test for an illness that may lead to the wrong conclusion that a person as antibodies for the illness. The type II errors measures the false negatives. In medicine, this could mean a mistakenly negative result of an antibody test with the wrong conclusion that there are no antibodies. In the spam example, emails mistakenly flagged as spam would be false positives, the spam emails that are missed would be false negatives.

7.2.3 Binary Performance Metrics

We can define performance metrics that summarize aspects of the performance of a hypothesis in a single statistical marker. There are many different performance metrics that all measure different aspects of the performance. The table below lists eleven such metrics.

Metric	Description	Definition
True positive rate, recall, sensitivity	Percentage of positive instances that are predicted correctly.	$TPR = \frac{TP}{TP+FN}$
True negative rate, specificity	Percentage of negative instances that are predicted correctly.	$TNR = \frac{TN}{TN+FP}$
False negative rate	Percentage of positive instances that are predicted incorrectly as negative.	$FNR = \frac{FN}{FN+TP}$
False positive rate	Percentage of negative values that are predicted incorrectly as positive.	$FPR = \frac{FP}{FP+TN}$
Positive predictive value, precision	Percentage of positive predictions that are predicted correctly.	$PPV = \frac{TP}{TP+FP}$
Negative predictive value	Percentage of negative predictions that are predicted correctly.	$NPV = \frac{TN}{TN+FN}$
False discovery rate	Percentage of positive predictions that are predicted incorrectly because they should be negative.	$FDR = \frac{FP}{TP+FP}$
False omission rate	Percentage of negative predictions that are predicted incorrectly because they should be positive.	$FOR = \frac{FN}{FN+TN}$
Accuracy	Percentage of correct predictions.	$accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
F1 measure	Harmonic mean of recall and precision.	$F_1 = 2 \cdot \frac{precision \cdot recall}{precision+recall}$
Matthews correlation coefficient (MCC)	Correlation between the prediction and the actual values.	$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(FP+FN)(TN+FP)(TN+FN)}}$

Since there is an abundance of metrics, the question is which metrics to use. The evaluation of eleven performance markers at the same time, many of which are strongly correlated, because they are all computed from the same four values from the confusion matrix, does not make sense. However, there is a logic behind these metrics, that can help with the decision which metrics should be used in a given use case.

The first four metrics are rates in relation to the actual values. They can be used to answer the question "how much of my positive/negative data is labeled correctly". The combination of TPR and TNR is very important as these two metrics cover two questions that are important for many use cases: how many of the positives are found and how many of the negatives are found correctly? Consequently, these are well suited to evaluate the type I error and type II error, which are important in many fields, especially medicine. The other two metrics, the FPR and the FNR are the counterparts and can be directly calculated from as $FPR = 1 - TNR$ and $FNR = 1 - TPR$.

The next four metrics are rates in relation to the predictions. They can be used to answer the question "how many of my predictions are labeled correctly". The difference to the first four metrics is that the quality of the results is measured in relation to the predicted classes, not the actual classes. Otherwise, these metrics are relatively similar to the first four metrics.

A common property of the first eight metrics is that they should never be used on their own, i.e., you must never use only one of these metrics as the single criterion for optimization. The reason is that the first four metrics only consider one column of the confusion matrix, the second four metrics consider only one row. This means that if only one of them is used, the other column/row is ignored. As a consequence, *trivial hypotheses* are sufficient to achieve optimal results. What this means is best explained with a simple example. Consider you only want to optimize the TPR and no other metric. A trivial hypothesis that predicts everything as positive would be perfect, because the TPR would be one. However, this model would not be helpful at all. To avoid this, you must ensure that you use metrics that use at least three of the four values in the confusion matrix.

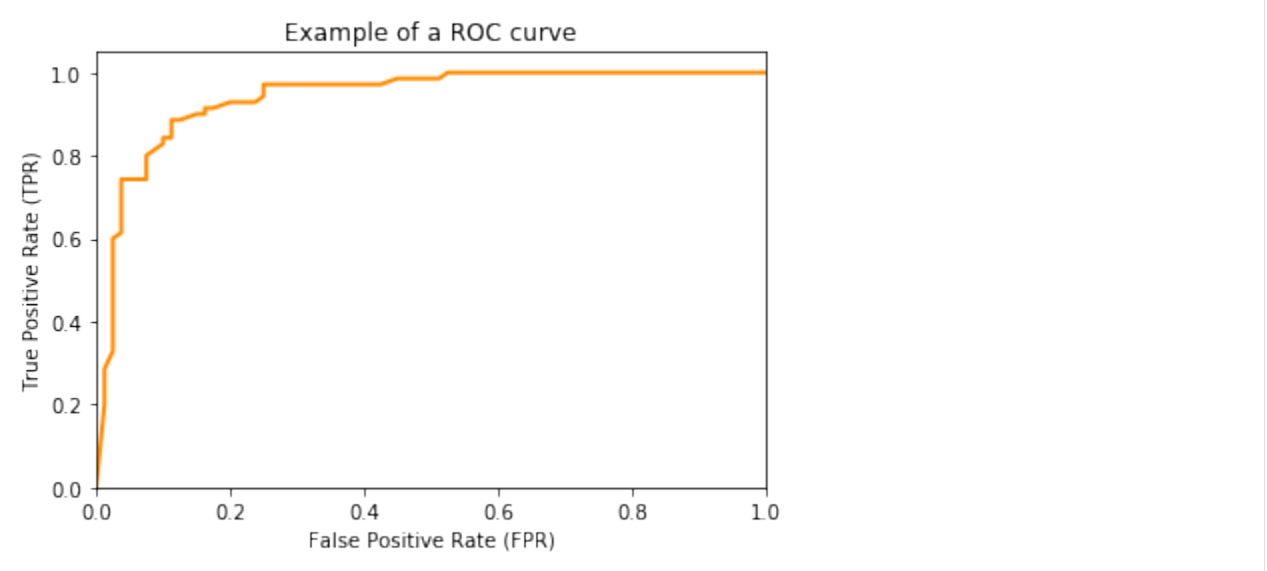
Alternatively, you may also use metrics that do not only evaluate a single aspect of the hypothesis, but rather try to evaluate the overall performance. The last three metrics are examples for such metrics. These metrics use the complete confusion matrix to estimate the performance of the classifier. The accuracy measures the percentage of prediction that are correct. This is similar to the first four metrics, but not for a single class, but rather for both classes at once. The drawback of the accuracy is that it may be misleading in case of *class level imbalance*. Class level imbalance is the problem when you have much more data of one class, than of the other class. For example, consider the case where you have 95% of data in the negative class and only 5% in the positive class. A trivial hypothesis that predicts everything as negative would achieve an accuracy of 95%, which sounds like a very good result. However, there is no value in such a model. Thus, accuracy should be used with care and only in case the classes are balanced, i.e., about the same amount of data from each class is available.

The F1 measure uses a different approach and is actually the *harmonic mean* mean of the TPR/recall and PPV/precision. Thus, it takes the ratio of of positive instances that are correctly identified and the ratio of positive predictions that actually should be positive into account. The harmonic mean is often used instead of the arithmetic mean for the comparison of rates. The F1 measure works well even in case of imbalanced data, because there is usually a trade-off between precision and recall assuming that a perfect prediction is impossible. To increase the recall, more positive predictions are required. However, in case the prediction is imperfect, this means that there will also be more false positives, which may decrease the precision. Thus, optimizing for the F1 measure is the same as optimizing for a good balance of true positives and false positives in case of imperfect predictions.

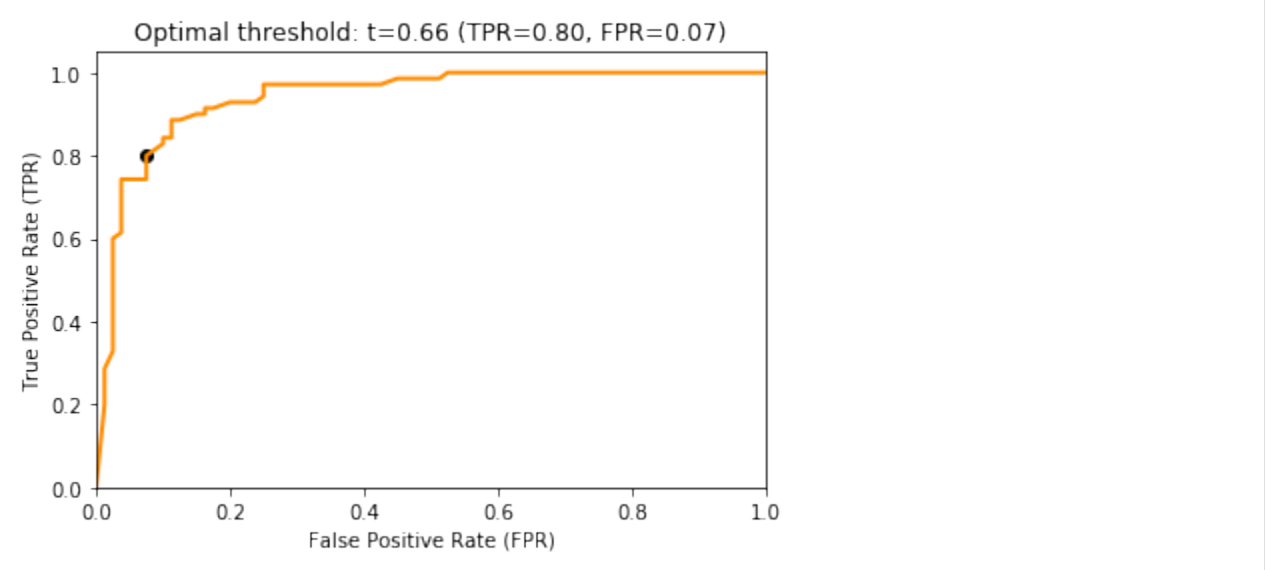
The last metric in the table above is MCC and directly measures the correlation between the expected results and predictions. Basically, the MCC measures if the rates of correct positive and negative predictions correlate well with the expected result. The MCC is very robust against class level imbalance and generally works very well. The main drawback of MCC is that the value cannot be easily interpreted. The metrics above all have an easy natural language explanation that can easily be explained to non-data scientist. MCC lacks such a simple explanation. This also makes it harder to evaluate how high MCC should be, in order for a hypothesis to achieve a sufficiently high performance. This is made worse because in contrast to all other metrics above, the values of MCC are not in the interval [0,1], but rather between [-1, 1], because it is a correlation measure. A high negative value means that our hypothesis does the opposite of what it should be doing. Depending on the context, this may mean that the hypothesis is very good, if you just invert all results of the hypothesis. Thus, both large negative values and large positive values are good. In conclusion, MCC is a very robust and informative metric, but requires more training for the correct use than the other metrics we discussed above.

7.2.4 Receiver Operator Characteristics (ROC)

All performance metrics we considered so far were based on the confusion matrix and did not account for scoring of classifiers. The drawback of the confusion matrix is that it can only be calculated for a fixed threshold t for a score-based hypothesis h'. How the values of the performance metrics change with different values for the threshold t is not accounted for. One way to consider how different thresholds affect the performance of a hypothesis are ROC curves. A ROC curve represents possible combinations of the TPR and the FPR for different thresholds t. For example, a ROC curve may look like this.

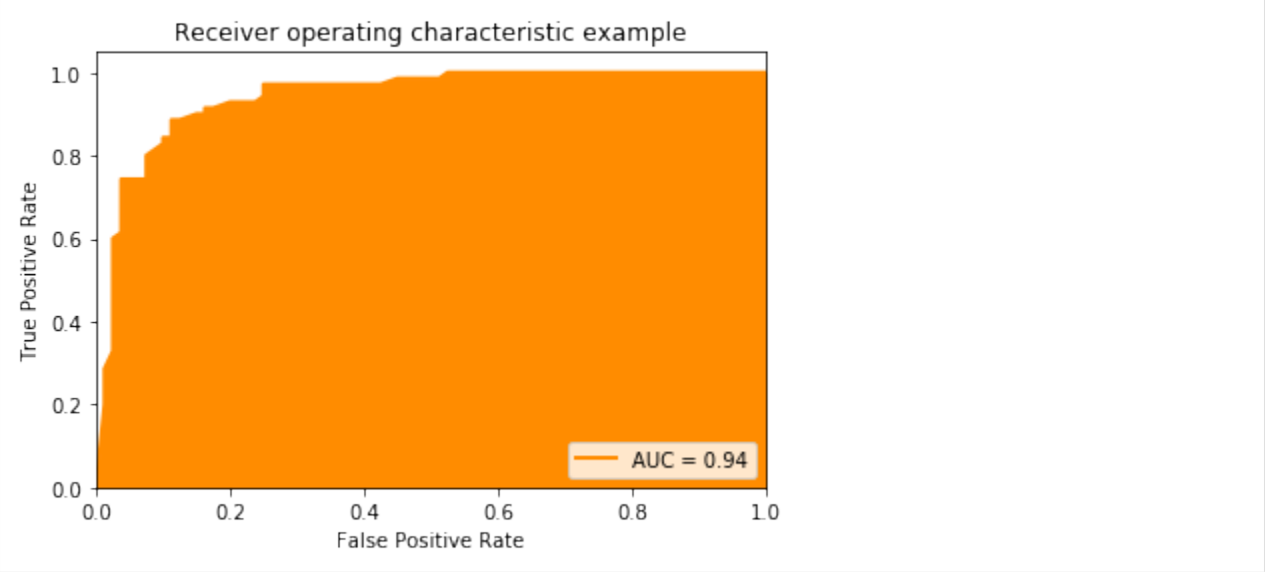


The ROC curve shows the FPR on the x-axis and the TPR on the y-axis. The ROC curve shows which different trade-offs between FPR and TPR are possible for different thresholds. Since good hypothesis achieve a high TPR and a low FPR, the optimal performance is in the upper-left corner of the plot where FPR is 0 and TPR is 1. This would be a perfect result without any misclassifications. The ROC curve can be used to pick a suitable combination of TPR and FPR for a use case. For example, if a TPR of at least 0.8 is required, we can see that we could achieve an FPR of 0.05, marked with a circle in the plot below.



7.2.5 Area Under the Curve (AUC)

The Area Under the Curve (AUC) is a performance metric based on the ROC curve. The ROC curve is a great tool for the visual analysis of the impact of different thresholds as well as for the selection of suitable thresholds if there is a desired goal for the TPR or FPR. The AUC computes a single value based on the ROC curve that estimates how well the hypothesis performs independent of any specific threshold. The idea behind the AUC is quite simple. If the best combinations of TPR and FPR are in the upper-left corner of the ROC curve, this means that the performance is better if the area under the curve increases. Thus, if we integrate the area under the ROC curve, we can estimate how well a classifier performs. Hence, the name: we have the area under the (ROC) curve.



The closer the AUC value is to one, the better the performance of our classifier. In terms of interpretation, AUC has similar problems as MCC: while a value of 1 is optimal, the worst value is not 0, but rather 0.5. This is because if we just randomly guess if we have a positive or a negative instance, the TPR equals the FPR, which is the diagonal of the ROC curve. Thus, an AUC value of 0.5 represents random guessing. A value of 0 means that we have a TPR of 0 and an FPR 1. Same as for negative values of MCC, we could just do the opposite of the hypothesis and get a perfect result. Consequently, values of AUC are better, the farther they are away from 0.5.

7.2.6 Micro and Macro Averages

All performance metrics we discussed so far are defined using the binary confusion matrix. While the concept of accuracy can be generalized to more than two classes, because it is the percentage of correctly classified classes, this is not easily possible with the other metrics. However, we can define multiple binary confusion matrices from a non-binary confusion matrix. Consider our confusion matrix with three classes from above.

		actual class		
		whale	bear	other
Predicted class	whale	29	1	3
	bear	2	22	13
	other	4	11	51

We can define three confusion matrices from this confusion matrix, in each of which one class is positive and all other classes negative. Here is the example with whale as positive class.

```
<tr><td></td><td colspan=3><b>Actual class</b></td></tr>
<tr><td rowspan=3><br><br><b>Predicted class</b></td><td><td><b>whale</b></td><td><b>
↪not whale</b></td></tr>
```

(continues on next page)

(continued from previous page)

```
<tr><td><b>whale</b></td><td>$TP_{whale} = 29$</td><td>$FP_{whale} = 5$</td></tr>
<tr><td><b>not whale</b></td><td>$FN_{whale} = 6$</td><td>$TN_{whale} = 97$</td></tr>
```

We can calculate all performance metrics from above now individually for all classes, e.g., the TPR for the class whale as

$$TPR_{whale} = \frac{TP_{whale}}{TP_{whale} + FN_{whale}}.$$

If we want to get performance metrics for all classes at once, we can use *macro averaging* and *micro averaging*. The macro average of a performance metric is the arithmetic mean of the metric applied individually for all classes. For example, for the TPR the macro average is defined as

$$TPR_{macro} = \frac{1}{|C|} \sum_{c \in C} \frac{TP_c}{TP_c + FN_c}.$$

With micro averaging, the performance metric is calculated directly by changing the formulas to use the sum of all classes. For example, the micro average is defined as

$$TPR_{micro} = \frac{\sum_{c \in C} TP_C}{\sum_{c \in C} TP_C + \sum_{c \in C} FN_C}.$$

Whether the macro average or the micro average is a better choice depends on the use case and the data. If the data is balanced, i.e., there is a similar amount of data for each class, the results for the macro average and the micro average will be almost the same. When the data is imbalanced, i.e., there are classes for which there are fewer instances than others, the macro average will treat all classes the same. Because the macro average is the arithmetic mean of the performance metrics for each class, the impact on the average is the same for all classes. This is not the case for the micro average, where classes with less instances have a smaller impact, because the count is lower and the individual counts are used.

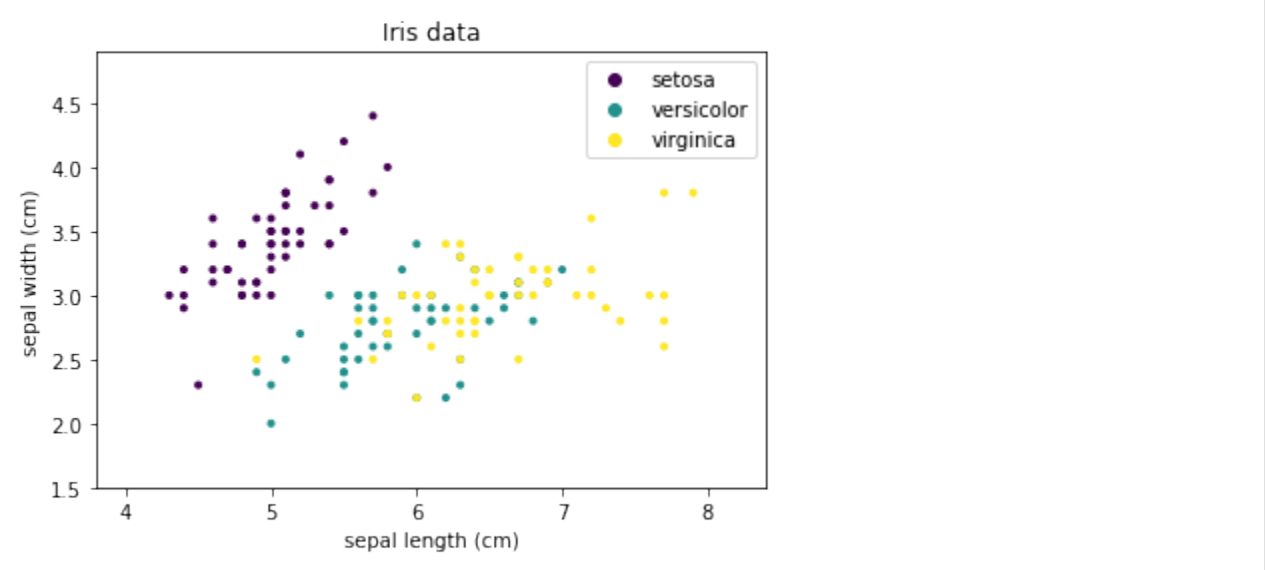
Thus, in case of class level imbalance, the macro average has the advantage that all classes are fairly represented in the average. The drawback of this is that classes with only few instances may have an overly large impact on the macro average. Vice versa, the impact on the micro average of each class is only as large as the number of instances for the class, but there is a risk that classes with few instances may be ignored.

7.2.7 Beyond the Confusion Matrix

All metrics above are defined using the confusion matrix. This is the standard approach for performance estimation in the machine learning literature. However, while such metrics are common and often have a relation to use cases, they all have one assumption, which is usually unrealistic: all errors are equal. In practice, some errors matter more than other errors. For example, lending a huge amount of money to a customer caries a greater risk than a small loan. If a customer defaults on a large loan, the impact of a false positive credit score is thus larger than for a small loan. Therefore, it is always a good idea to think about concrete costs, benefits and risks associated with the true positives, false positives, true negatives, and false negatives and ideal define a cost matrix that specifies the individual gains and losses for each of these cases. This way, you can define a customized cost function for use cases, which usually means that you get more meaningful results, e.g., demonstrated here.

7.3 Decision Surfaces

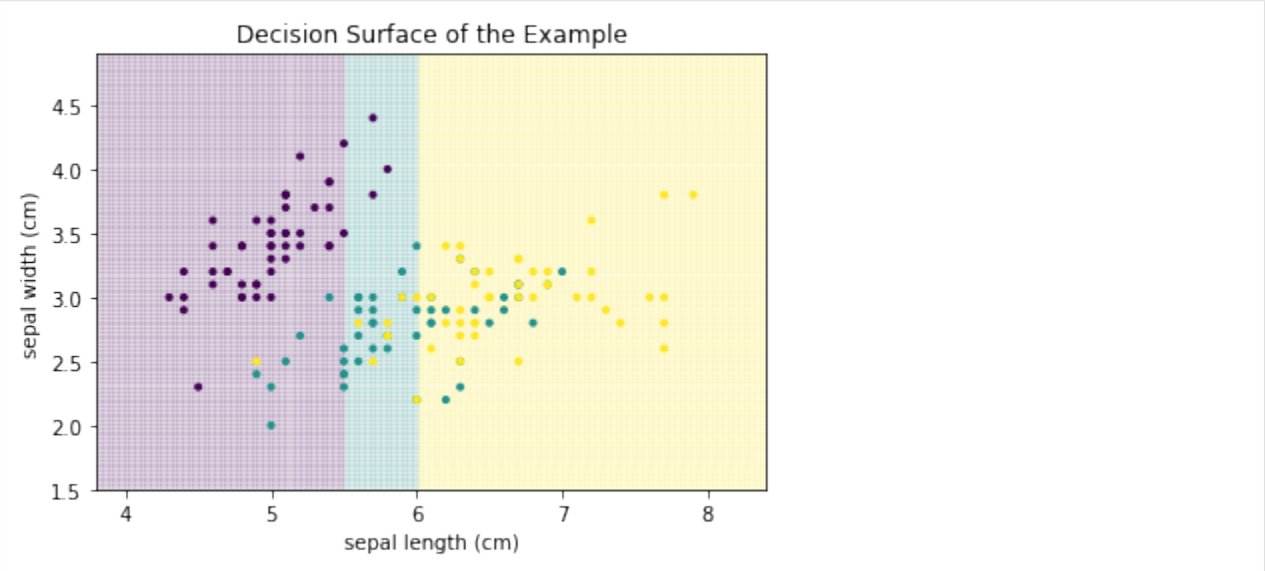
We will use a visual aid to demonstrate how different algorithms decide how data is classified: the *decision surface*. The decision surface uses colors to show the class of data. The drawback of decision surfaces is that they can only be (easily) used with 2-dimensional data. We will use the sepal length and sepal width of the Iris data as sample data to train classifier.



We can see that Setosa is clearly separated, while there is overlap between Versicolor and Virginica. We will colorize the background of the above plot with the classes that the classifiers predict to visualize the decision surface: purple background for Setosa, teal for Versicolor, and yellow for Virginica. We demonstrate this with a simple example:

- All instances with sepal length less than 5.5 are predicted as Setosa.
- All instances with sepal length between 5.5 and 6 are Versicolor.
- All instances with sepal length greater than 6 are Virginica.

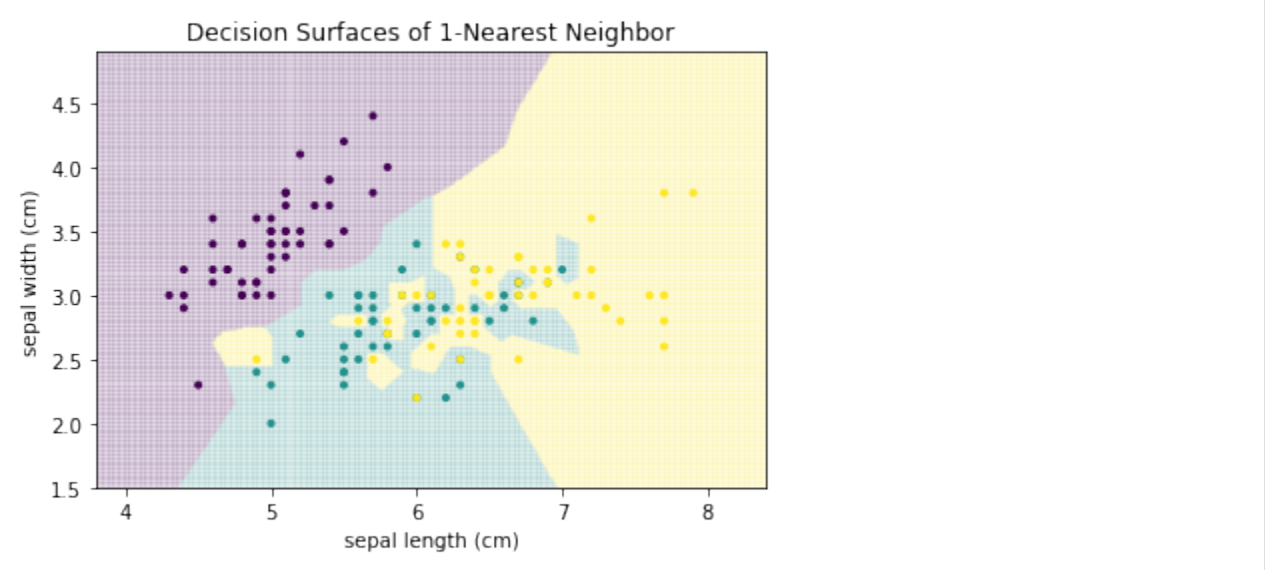
This classifier would result in the following decision surface.



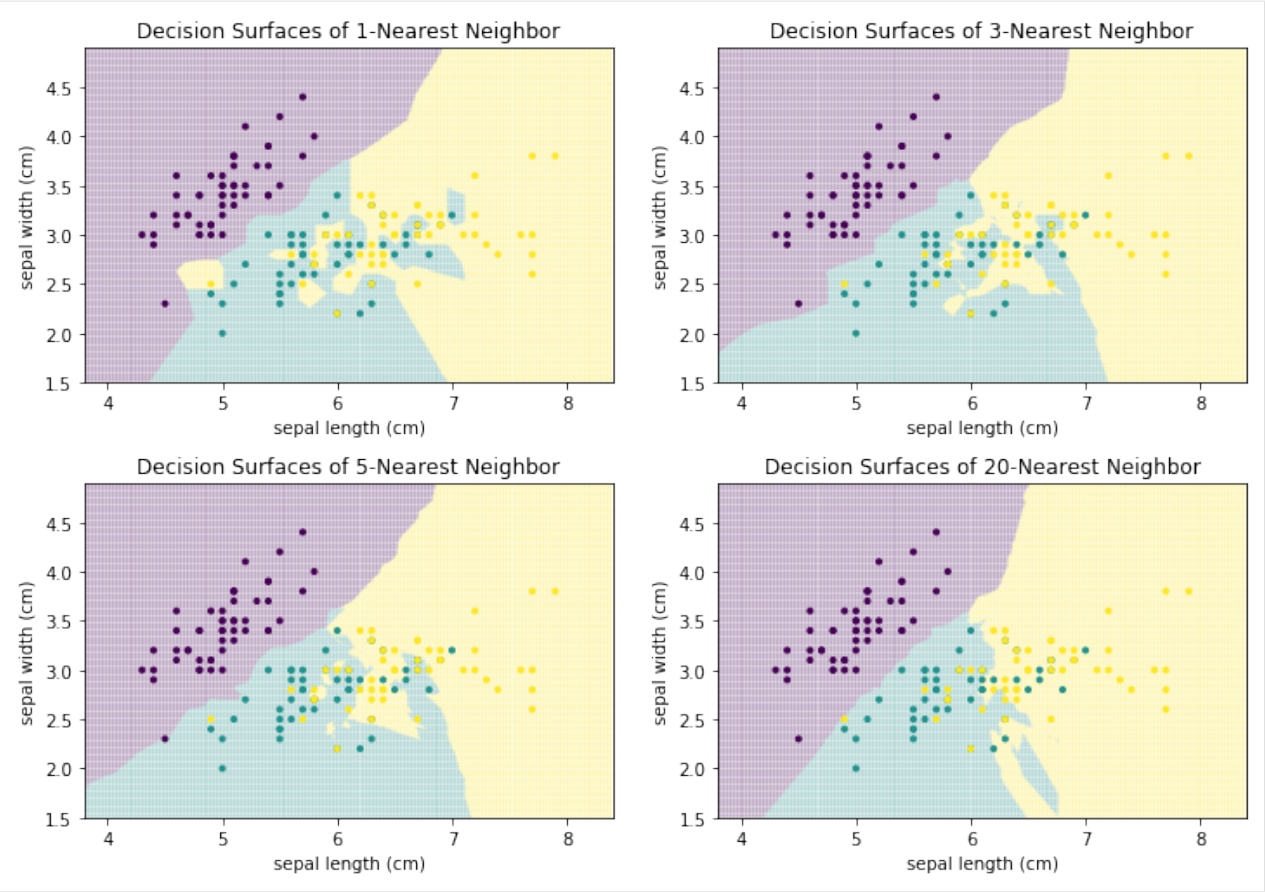
The line where the color changes is called the *decision boundary*.

7.4 k-Nearest Neighbor

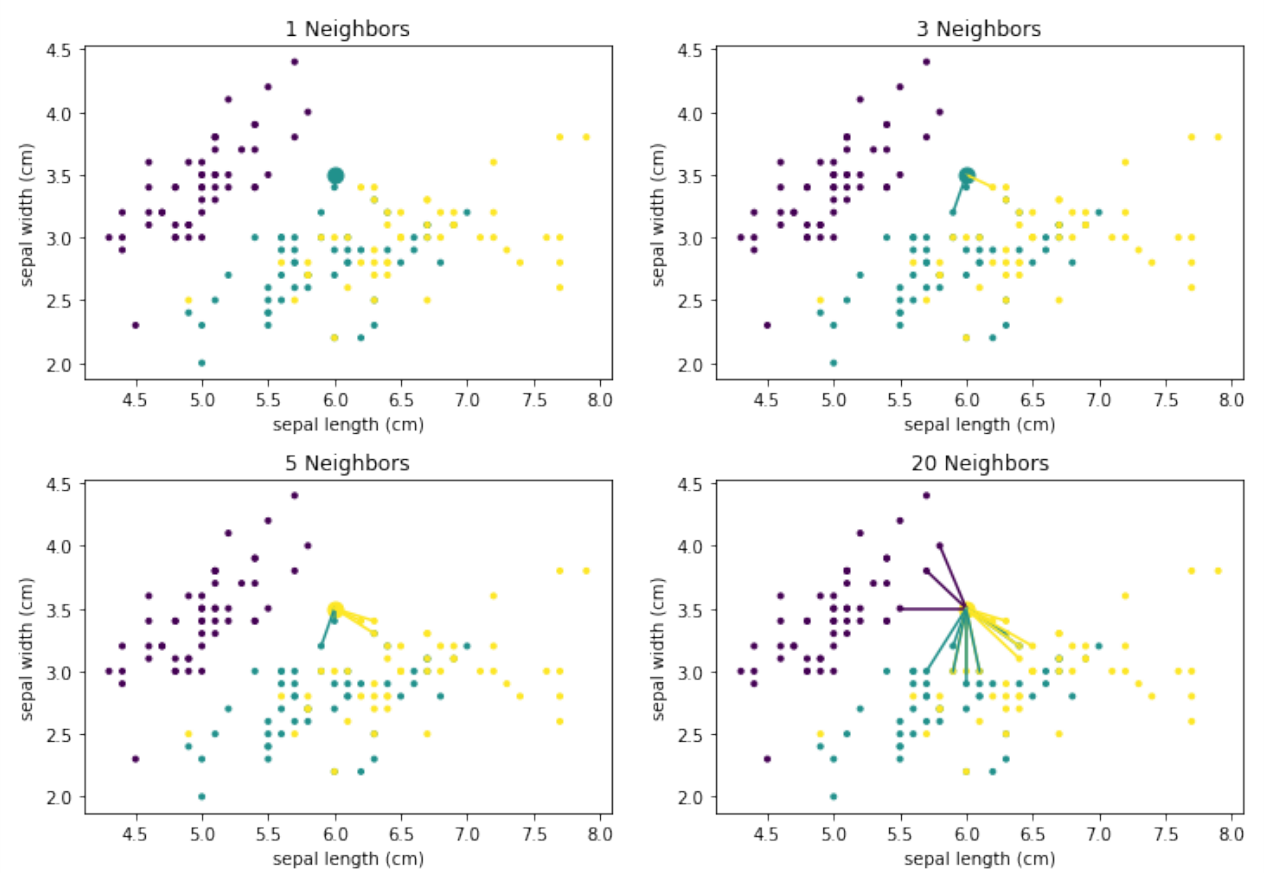
The first classification algorithm we consider is based on the idea that instances of the same class are close to each other. This is similar to the intuition behind clustering algorithms that use the distance to estimate the similarity of instance. The simplest approach is that each instance is classified based on the nearest neighbor, i.e., the instance in the training data that is closest. For the Iris data, this would lead to the following decision surface.



We can generalize this concept to k-Nearest Neighbors: the class is the result of voting between the k instances that are closest to the instance where we want to predict the class. The following figure shows how the decision surface changes with different values for k.

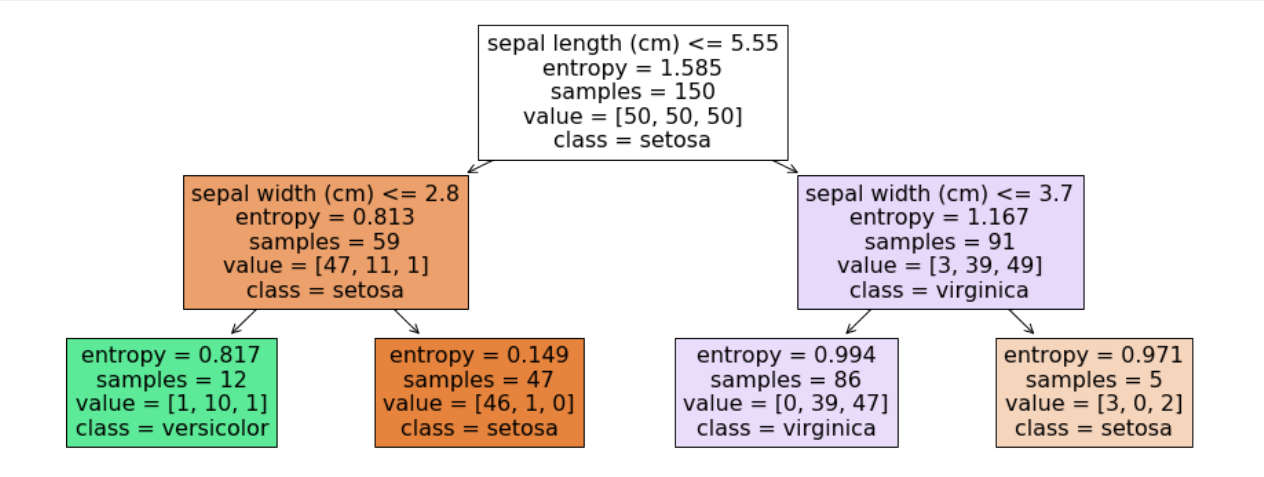


We can see that there is no clear structure in the decision surface of the k-Nearest Neighbor classifier. The decisions boundaries have many sharp edges and do not look regular, e.g., like they are the result of a differentiable curve. This is because k-Nearest neighbor does not calculate a hypothesis that is some mathematical generalization of the data. Instead, we have an *instance-based* algorithm, which defines the decision boundaries by direct comparisons between instances. When we compare how the result evolves with larger values of k we see that single points have a smaller influence. For example, with $k = 1$, there is a single yellow instance on the left side of the plot, which looks like an outlier. This outlier leads to a relatively large area that is yellow, where this region should clearly be rather purple or teal. With larger neighborhood sizes, this effect vanishes. On the other hand, large neighborhood sizes mean that instances that are further away can participate in the voting. With $k = 20$ this leads to a relatively sharp boundary between teal and yellow, because not the points close to the boundary have the largest influence, but rather the points behind them. As soon as there are more yellow than teal points, the decision surface stays yellow. In comparison, with the smaller neighborhood size there are "islands" of teal within the yellow area. The following plots show how the neighborhood of the point (6,3.5) changes, which also means that the classification changes from yellow to teal.



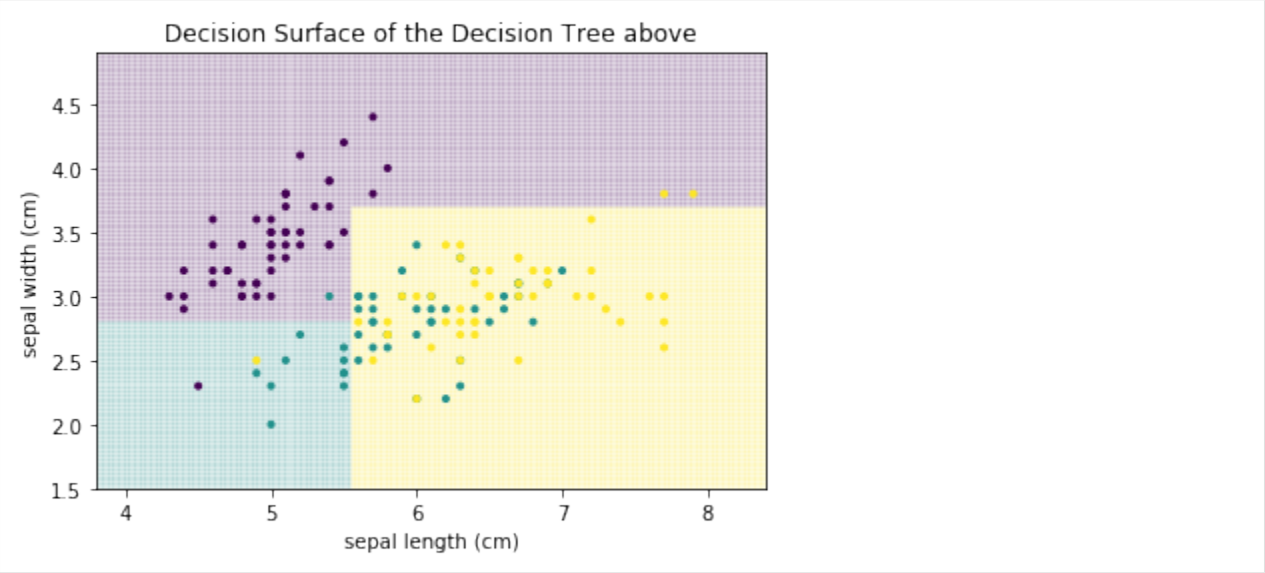
7.5 Decision Trees

Imagine trying to explain why you want to buy a certain car. You are likely describing features of the car that were your criteria for the decision: a car with at least five doors, a certain amount of horse power, etc. Some of these criteria are more important than others, i.e., you apply them first for selecting your car. This is the intuition behind a decision tree: logical rules are applied to features to make decisions about the class of instances. These decisions are structured in a tree as follows.



In the example, the first decision is made about the sepal length. If it is less than or equal to 5.55, we go to the left side of the tree, if if is greater than 5.55, we go the the right side of the tree. We also see more information in the nodes of the tree. We ignore the entropy for now. Samples are the number of instances during the training, on which this decision was learned. Value are the number of instances of each class within the samples. In the first node (which is also called the root node), we have 150 samples, 50 of each class. The decision tree then just decides on the first class observed in the data, which is Setosa in this case. The data is then partitioned based on the logical rule. 59 instances, 47 of which are Setosa have a sepal length less than or greater to 5.55, 91 instances have a sepal length greater than 5.55, 39 of which are Versicolor, 49 are Virginica. This shows the value of the decision trees: not only do we understand how decisions are made about the features, we also see the effect of the decisions on the data. The nodes on the lowest layer of the tree are called leaf nodes.

The decision surface of the above tree looks as follows.



We can see that the decision boundaries that separate the decision surface between the classes are parallel to the axes. This is a general property of decision trees: because we have a logical decision, usually of the form leq or geq, the resulting partitions of the data are always cutting the current partitions in half in orthogonal to the axis of the feature and parallel to all other axes.

The general idea how decisions trees are learned is based on a relatively simple recursive algorithm:

1. Stop if the data is "pure", the amount of data is "too small", or the maximum depth is reached.
2. Determine "most informative feature" X'
3. Partition training data using X'
4. Recursively create subtree for each partition starting with step 1.

Thus, the general idea is to find the "most informative feature", partition the data using this feature, and repeat this until the data is "pure" or there is not enough data left in the partition and the data is "too small". The concept of "too small" is always the same: if the number of instances is below a certain threshold, the recursion stops. A similar criterion is the maximum depth. This can be used to define a hard limit on the complexity of the tree. The depth of the tree is defined as the maximum number of decisions that are made, before the class is determined. For the purity and the most informative features, there are different concepts. Which concept is used and how exactly the partitions are determined depends on the variant of the decision tree, e.g., CART, ID3, or C4.5.

Here, we want to briefly present how these concepts can be implemented based on *information theory*. The idea is to describe purity as the uncertainty in the data and the most informative feature based on the *information gain* if this feature is known. Information theory deals with the uncertainty of random variables. Therefore, we consider everything as random variable: the classification of instances is a discrete random variable C, our features are random

variables $X_1, ..., X_m$. The core concept of information theory is the entropy of random variables. The higher the entropy, the more random a decision. For example, the entropy of flipping a fair coin that has a fifty percent probability for both heads and tails is 1. The entropy for an unfair coin that always lands on heads is 0. Thus, if we want to make informed decision with a high certainty, we want to find partitions of the data which minimize the entropy of the classification C. The definition of this entropy is

$$H(C) = -\sum_{c \in C} p(c) \log p(c)$$

where $p(c)$ is the probability of observing the class c in the partition. If this entropy is below a defined threshold, the data is "pure".

The *conditional entropy* tells us the uncertainty of the classification, if we have perfect information about a feature and is defined as

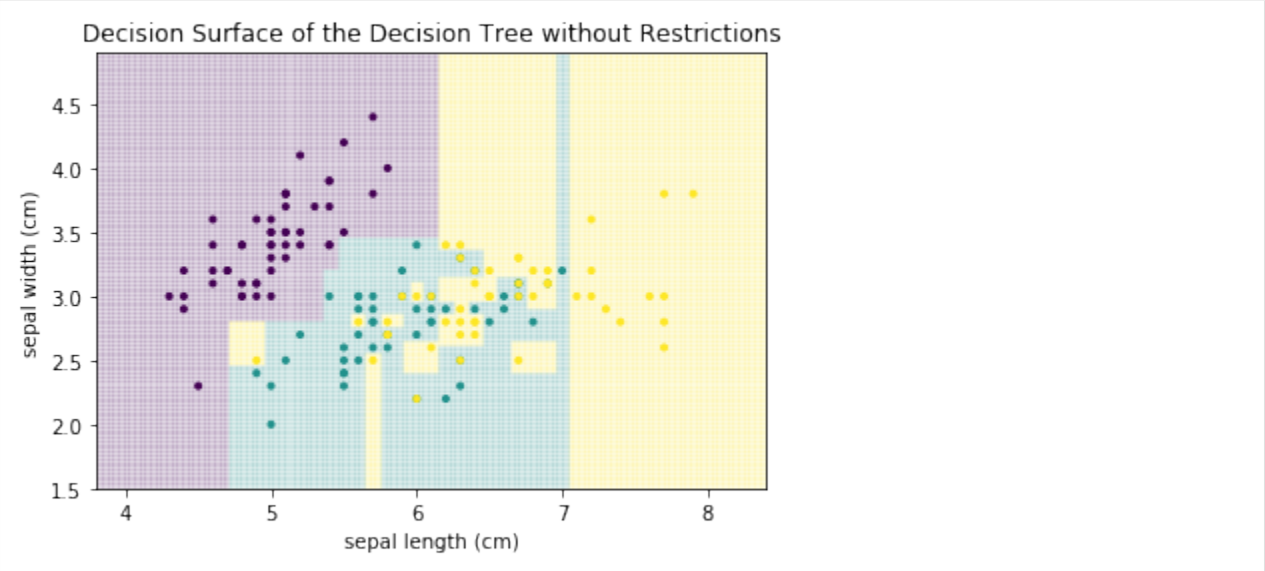
$$H(C|X') = -\sum_{x \in X'} p(x) \sum_{c \in C} p(c|x) \log p(c|x)$$

where $p(c|x)$ is the probability of observing the class c given the value x of the feature $X' \in \{X_1, ..., X_m\}$. Thus, the conditional entropy is a measure for how much information about the classification is contained in the feature X'. The lower the conditional entropy, the smaller the uncertainty about the classification if the feature is known. Thus, features are more *informative*, if the conditional entropy is smaller. When we combine the entropy of the classification and the conditional entropy of the classification if a feature is known, we get the *information gain* as the reduction in uncertainty, i.e.,

$$I(C; X') = H(C) - H(C|X').$$

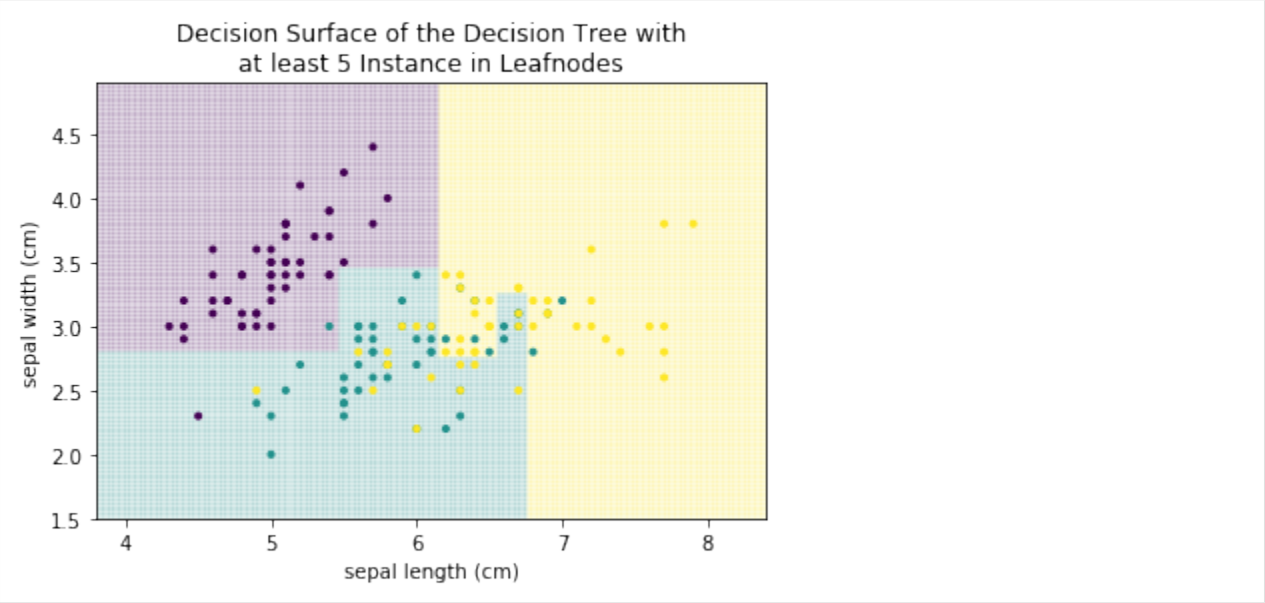
Thus, the most informative feature is the feature that maximizes the information gain. Once the feature is determined, the algorithms decide on a logical rule to create the partitions.

The example of the decision surface above was for a very low depth, i.e., we only allowed two decisions. While this is useful to display the structure of the decision tree and explain how the decision surface looks like, most problems require decision trees with greater depth. If we do not limit the depth and define what "too small" means, we get a decision tree with the following decision surface.



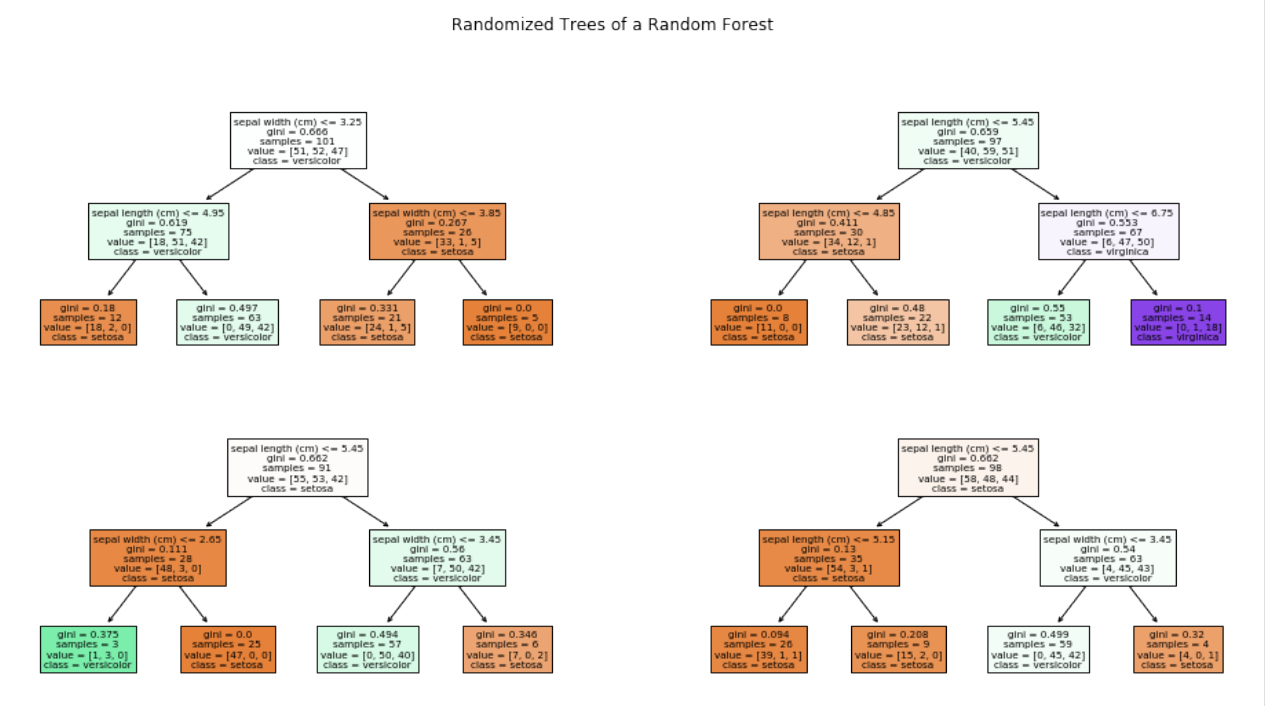
We can see that there are many more decisions. More importantly, we can see that *overfitting* occurred. For example, there is a small teal area at the sepal length 7, that cuts right through the yellow area, just because there is a single teal instance. Similarly, there is are several very small yellow rectangles in the center of the plot that cut single points

out of the teal area. These are all typical examples of overfitting. Decision trees tend to overfit data, if the number of decisions (i.e., the depth) and/or the minimal number of samples required in a partition is not limited. If we limit the tree such that we allow only leaf nodes that have at least 5 instances, we get the following result.

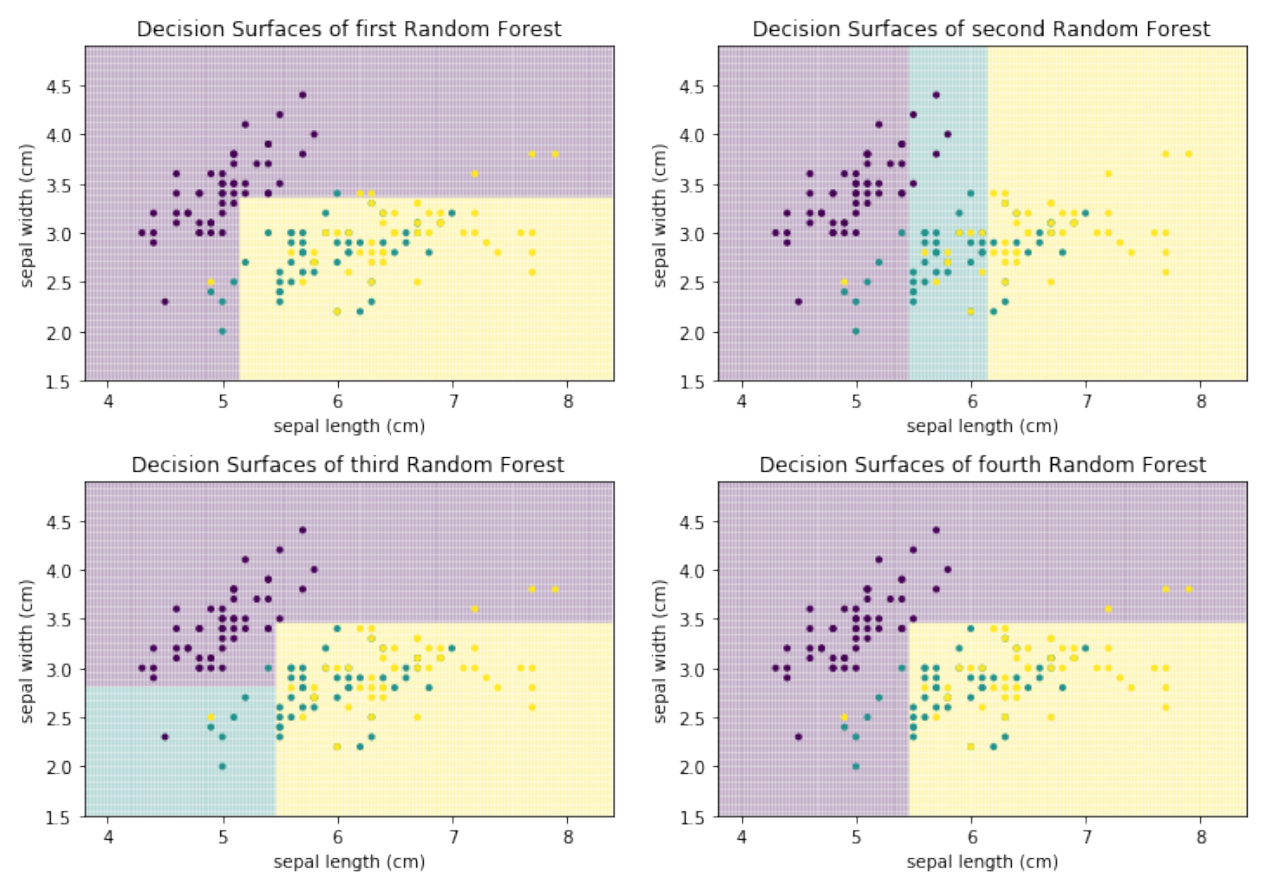


7.6 Random Forests

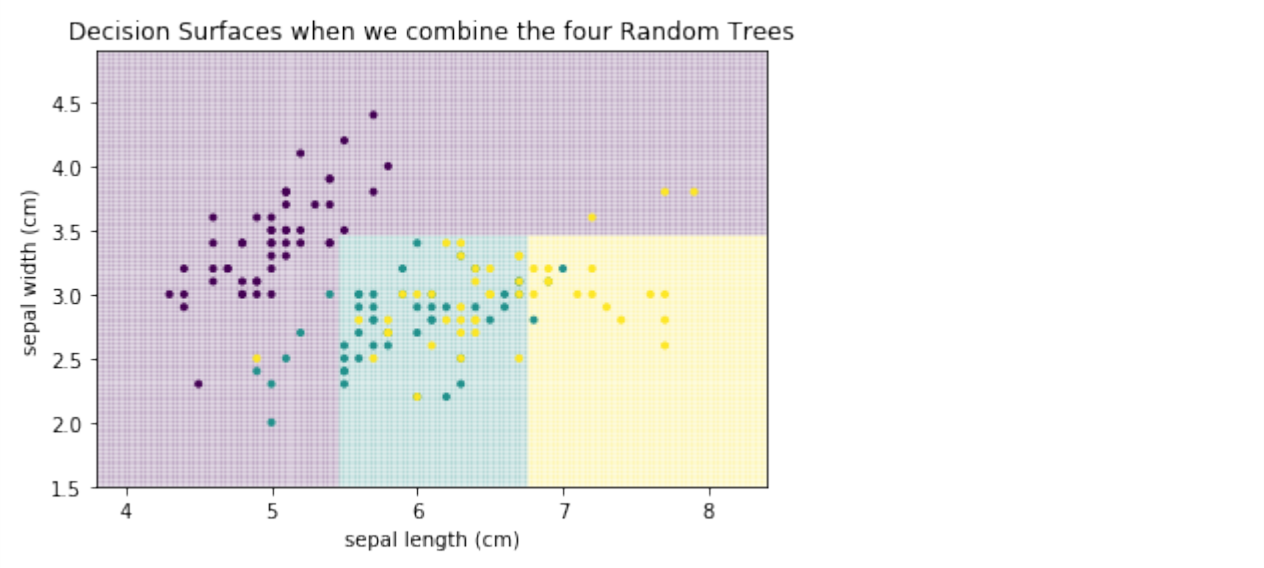
The general idea behind random forests is to combine multiple decision trees into a single classifier. This is known as *ensemble* learning, i.e., a random forest is an ensemble of decision trees. Below, you see how a random forest may be created from four decision trees.



Each of these decision trees on its own is bad. This is further highlighted when we look at the decision surfaces of the four trees.

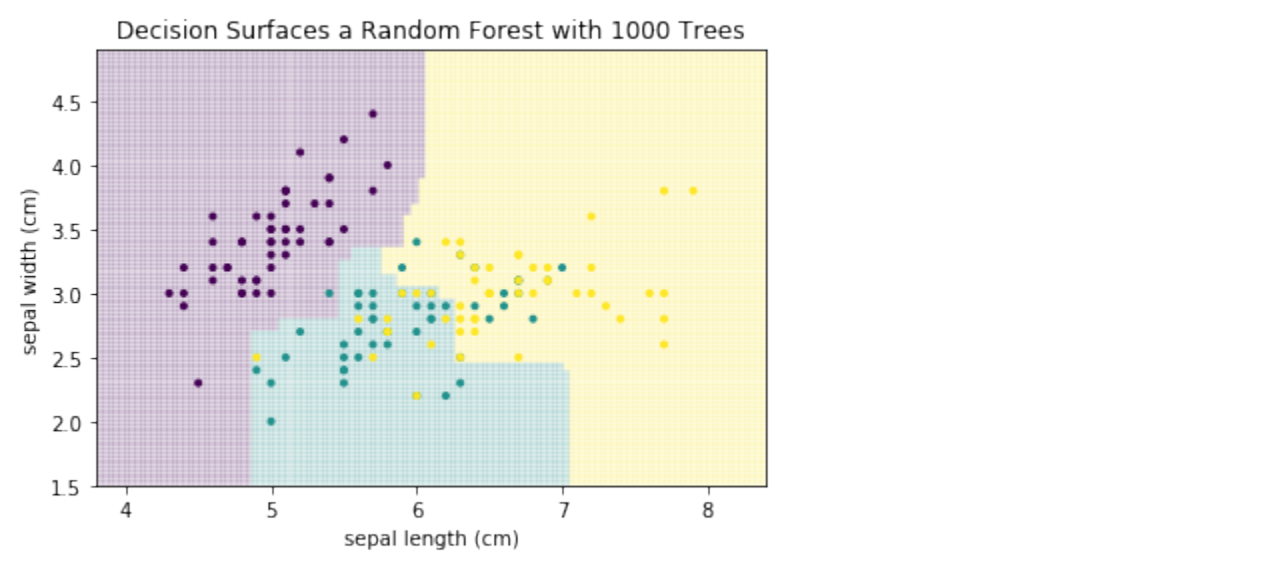


For example, the first and fourth decision tree both never classify anything as Virginica (teal). Thus, individually, we have four *weak classifiers*. However, if we combine the results of the four classifiers and output the average prediction of each tree, we get the following.



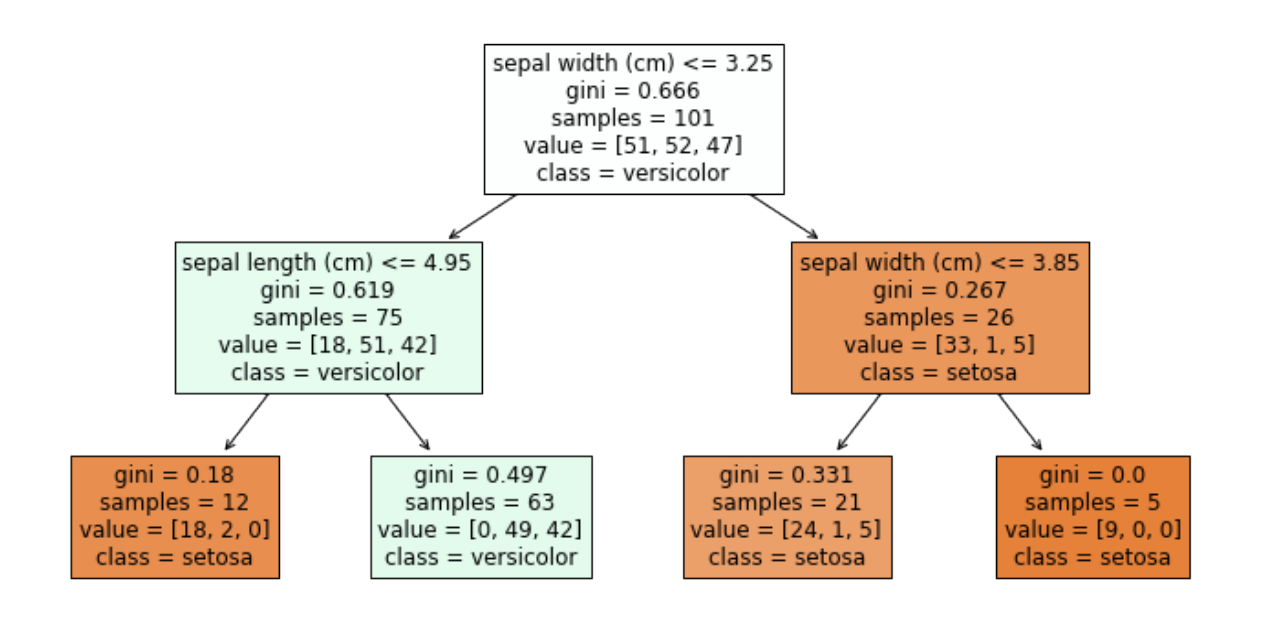
This is a much better result, even though this is only based on four weak classifiers. This concept, to build a *strong*

classifier by averaging over many weak classifiers is comparable to the audience joker of Who Wants to be a Millionaire. Individually, the audience is a weak help. However, because the questions are in a way that often most people guess the correct answer, the average prediction of the group is reliable. The same is true in this example. Individually, the four decision trees are bad. However, some are good at identifying purple, others at yellow. Taken together, the individual strengths add up and cancel out the weaknesses. Usually, this is not done with four trees, but with hundreds, or even thousands of trees. Below, you can find the result with 1000 trees.



The explanation of random forest so far is only for the *ensemble* nature of the approach, i.e., how a strong classifier is created from many weak classifiers. The open question is, how does the random forest determine different decision trees from the same data, that are used as weak classifier? In general, the algorithms for the training of decision tree are deterministic, i.e., if you run the algorithm twice on the same data, you get twice the same decision tree. Therefore, we must explore how the training of decision trees is randomized by random forests.

We get our first indicator for this, when we look at the individual weak learners. Here is the first of the four decision trees from above.



We observe that the number of samples in the first node is 101 and we have 51 Setosa, 52 Versicolor, and 47 Virginica. This means that this is not our original Iris data. What we observe is the first randomization of the training of the decision trees in random forests: the training data is a so-called *bootstrap sample* of the original data. This means that if we have 150 instance in our training data, we randomly draw with replacement 150 instances from our training data. Because we draw with replacement, we will have some instances multiple times, and other instances will not be part of the bootstrap sample. On average, we expect that we draw about 63.2% unique instances from the original data, the remaining data will be duplicates. Thus, all decision trees in a random forest get different instances for the training. Drawing bootstrap samples to train multiple classifiers as an ensemble is also called *bagging*, which is short for *bootstrap aggregating*.

The second randomization of random forest is a randomization of the features. Not every decision tree gets all features for the training. Instead, a random subset of the features is used. Commonly, the square root of the number of features is used, e.g., if we have four features, each tree may only use $\sqrt{4} = 2$ features. The rationale for the subsets of features is that otherwise there is a large risk that all trees would use the same small subset of most informative features and other, less informative features, would not play a role for the classification. This would go against the idea of the ensemble, which is to use many different weak learners with different strengths. If all weak learners would reason over the same (or very similar) features, there would be a high risk that the weak learners would all have the same strength and weaknesses.

The risk of always using the same small subset of features is amplified by another aspect of random forests: often, the individual decision trees have a very low depth. This makes sense, because we want weak learners. However, this means that every tree uses only few features, which increases the risk that we would always have a similar set of features over which the decisions are made. The randomization of the available features counters this problem effectively and gives every feature a chance.

7.7 Logistic Regression

Logistic regression tries to estimate the *odds* that an instance belongs to a class. To understand what this means, we briefly revisit the concept of odds from statistics. Let $P(Y = c)$ be the probability that a random variable Y equals c. The odds of Y being c are defined as

$$odds(c) = \frac{P(Y = c)}{1 - P(Y = c)}.$$

What this means gets clear with an example. Assume the random variable Y models the probability of passing an exam and $c = pass$ with $P(Y = pass) = 0.75$. This means that the odds of passing the exam are

$$odds(pass) = \frac{0.75}{1 - 0.75} = 3.$$

In other words, the odds of passing the exam are three to one.

The odds are closely related to the logistic or *logit* function, which is defined as

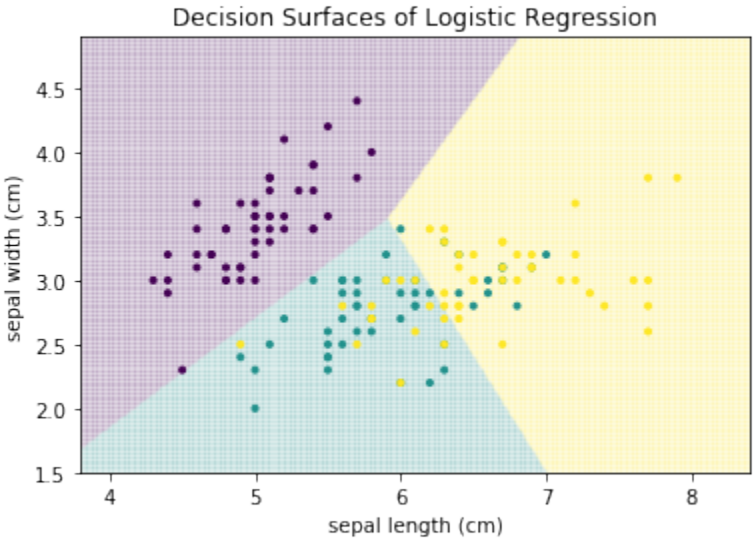
$$logit(P(Y = c)) = \ln \frac{P(Y = c)}{1 - P(Y = c)},$$

i.e., as the natural logarithm of the odds of c. This function is the namesake of logistic regression. When we say that the random variable Y models that probability that an instance belongs to a class, and $c \in C$ are our classes, then $logit(P(Y = c))$ is nothing else but the logarithm of the odds that the instance is of that class.

The regression is nothing more than a linear regression (details in Chapter 8) of the form

$$logit(P(Y = c)) = \ln \frac{P(Y = c)}{1 - P(Y = c)} = b_0 + b_1 x_1 + ... + b_m x_m$$

for with features $\mathcal{F} = \mathbb{R}^m$. In two dimensions, the forumula of linear regression describes a line, in three dimensions a plane, etc. This is also how our decision boundaries look like.



The decisions are made along lines that separate the feature space into different regions. A strength of logistic regression is that we can understand how each feature affects the classification. This is because we have a regression model of the logit function. We can reformulate this using the exponential function as follows:

$$\begin{aligned} & \ln \frac{P(Y=c)}{1-P(Y=c)} = b_0 + b_1 x_1 + ... + b_m x_m \\ \Rightarrow \quad & \frac{P(Y=c)}{1-P(Y=c)} = \exp(b_0 + b_1 x_1 + ... + b_m x_m) \\ \Rightarrow \quad & odds(c) = \exp(b_0) \cdot \Pi_{i=1}^{m} \exp(b_i x_i) \end{aligned}$$

Thus, the odds of the class are the product of these exponentials. The impact of feature $i, i = 1, ..., m$ on the odds is $\exp(b_i x_i)$. From this follows that $exp(b_i)$ is the *odds ratio* of feature i. The odds ratio defines by how much the odds change due to this feature. For example, an odds ratio of 2 means that the odds would double if the value of the feature would be increased by one. In general, an odds ratio greater than one means that the odds increase if x_i increases, an odds ratio of less than 1 means that the odds decrease if x_i increases. Because the exponent of the coefficients is the odds ratio, the coefficients are also called the *log odds ratios*.

We can also determine the odds ratios for our logistic regression model of the Iris.

```
            sepal length (cm)  sepal width (cm)    intercept
setosa               0.066610         10.216701  2733.180675
versicolor           1.845467          0.207923     6.328398
virginica            8.134952          0.470746     0.000058
```

We can see that the odds of Setosa increase strongly with the sepal width and decrease with the sepal length. Similarly, we can see that the odds of Versicolor and Virginica both increase with the sepal length and decrease with the sepal width. The intercept are the odds in case all features are exactly 0, i.e., b_0. Thus, all changes due to increasing feature values are with respect to the intercept. Because the intercept of Versicolor is larger than that of Virginica, there is a region in which the odds for Versicolor are larger, even though the increase in odds due to the sepal length is weaker and the decrease in odds is stronger than for Virginica.

> Note:
>
> In this case the intercept is misleading. While the intercept for Setosa is very high, the coefficient for sepal length is very low. Given that the instances all have sepal lengths greater than four, this means that the impact of the large intercept vanishes because of this. For an in-depth analysis, the data should always be

centered, e.g., with Z-score standardization. Moreover, in case the interpretation of the coefficients is the primary goal, you should consider to use a package like statsmodels instead of scikit-learn, because the regression analysis is more detailed, especially with respect to the statistical significance of the results.

7.8 Naive Bayes

Bayes Law is one of the fundamental theorems of stochastics and defined as

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}.$$

If the random variable X are our features, and the random variable Y are our classes, $P(Y|X)$ is the probability of a class given the features. A perfect estimation of this would be an ideal scoring function for a score based classifier. In our case, we have

$$P(c|x_1, ..., x_m) = \frac{P(x_1, ..., x_m|c)P(c)}{P(x_1, ..., x_m)}$$

for a class $c \in C$ and an instance $(x_1, ..., x_m) \in \mathcal{F}$. Thus the probability of the class c given the instance $(x_1, ..., x_m)$ is the probability of this instance given the class multiplied with the probability of observing this class and divided by the probability of observing the instance. The problem with Bayes Law is that it is usually very hard to accurately calculate $P(x_1, ..., x_m|c)$ and $P(x_1, ..., x_m)$, because this would require either the exact knowledge of the underlying joint probability distribution of all features, or an enormous amount of data, such that there are multiple observations for each possible instance.

Instead, we go from Bayes Law to Naive Bayes. The core of Naive Bayes is the "naive assumption" that the features are conditionally independent given the class. For example, for the Iris data this would mean that the sepal length of Setosa is independent of the sepal width of Setosa. The data clearly shows that this is not the case. Hence, this assumption is almost never fulfilled and, therefore, naive.

However, from a mathematical point of view, conditional independence given the class means that

$$P(x_1, ..., x_m|c) = \Pi_{i=1}^{m} P(x_i|c),$$

i.e., we do not need the joint distribution of all features anymore, but can work with the distribution of the individual features instead, which is much less complex. When we substitute this into Bayes Law, we get

$$P(c|x_1, ..., x_m) = \frac{\Pi_{i=1}^{m} P(x_i|c)P(c)}{P(x_1, ..., x_m)}.$$

We still have the joint probability $P(x_1, ..., x_m)$ in the denominator. However, this probability is independent of the class c. Since we are not really interested in the exact probability, but only in a scoring function that we can use for classification, we can just leave out the denominator, because it does not affect which class scores highest for an instance. Thus, we get

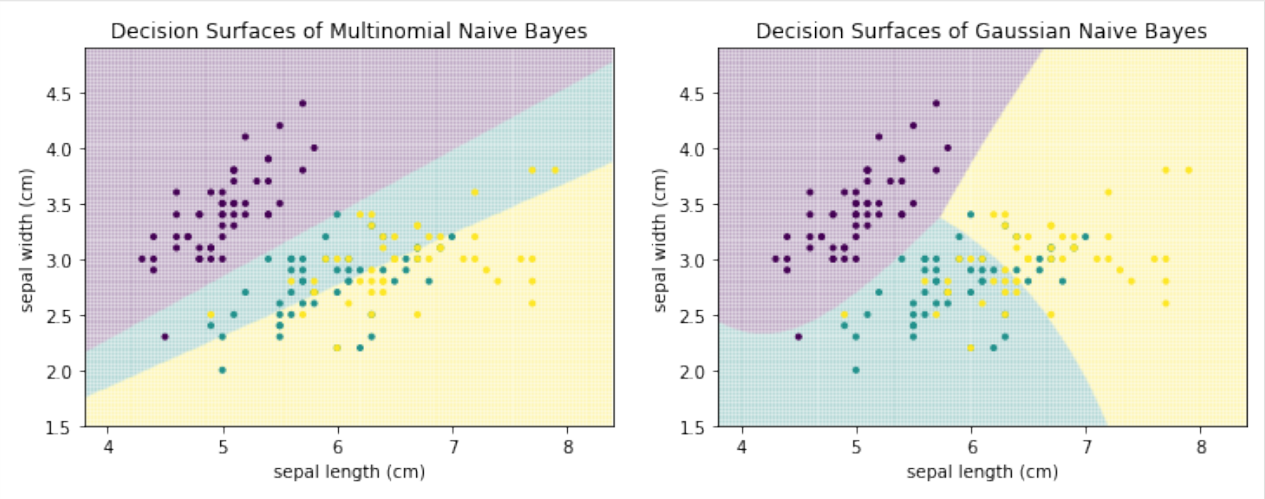
$$score(c|x_1, ..., x_m) = \Pi_{i=1}^{m} P(x_i|c)P(c).$$

We can calculate this scoring function and assign the class with the highest score. The open question is how $P(x_i|c)$ and $P(c)$ are calculated. For $P(c)$ this is straightforward, as this can be estimated by counting how often each class is observed in the data.

The two most popular approach for the estimation of $P(x_i|c)$ are *Multinomial Naive Bayes* and *Gaussian Naive Bayes*. Multinomial Naive Bayes calculates $P(x_i|c)$ by counting how often the value x_i is observed for the class c. This approach works well with categorical data and counts. Multinomial Naive Bayes does not work well with continuous numeric data, because the counts are very often exactly 1, because it is unlikely to observe the exact same value multiple times with continuous data. Gaussian Naive Bayes is better suited for such data. Gaussian Naive Bayes

assumes that each feature follows a normal distribution and estimates the probability $P(x_i|c)$ through the density function of a normal distribution that is fitted based on all instance values for a feature.

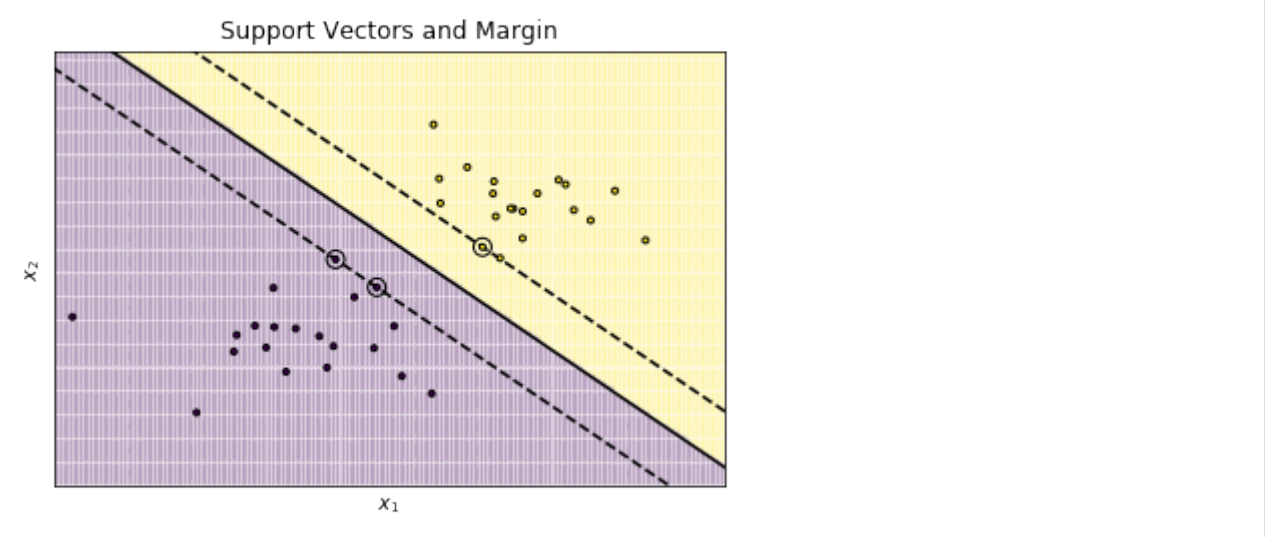
Below, we can see the difference between Multinomial Naive Bayes and Gaussian Naive Bayes for the Iris data.



The decision boundaries of Multinomial Naive Bayes are linear, whereas the decision surfaces of Gaussian Naive Bayes are quadratic. We can see that Multinomial Naive Bayes does not work well, especially the separation between green and yellow does not seem reasonable. This is expected, because the data is numeric. Gaussian Naive Bayes yields better results.

7.9 Support Vector Machines (SVMs)

Support Vector Machines (SVMs) are an approach based on the optimization of the decision boundary. We only discuss the general concepts of SVMs, the complete mathematical description of how and why SVMs work is beyond our scope. The following visualization will help us to understand how SVMs determine the decision boundary.



The plot shows two groups: the dots in the yellow and purple areas. The data can easily be separated with a single line. Three such lines are shown in the plot. The solid line is as far away from the data as possible, the two dashed lines are so close to the data, the they actually intersect with data points. For the data depicted in the plot, the classification with all three lines as decision boundary would be the same. The difference between the three lines is the *margin*, i.e.,

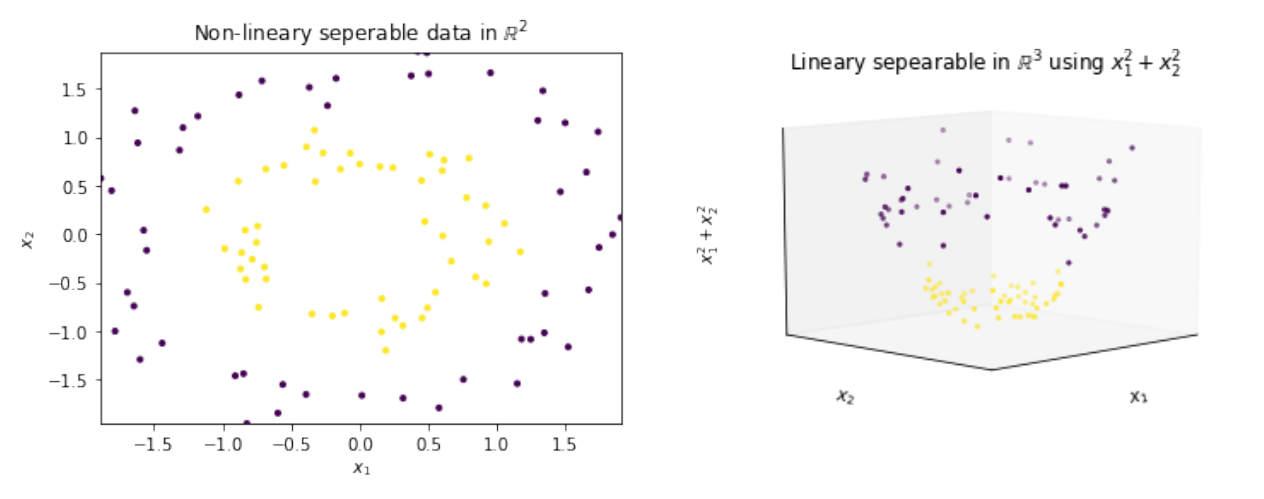
the minimal distance from the line to any instance of each color. The dashed lines *minimize* the margin, i.e., there is no gap between the decision boundary and the observed data. The solid line *maximizes* the margin, i.e., this is the line that has the largest possible distance to yellow and the purple instances. Moreover, the distance to the closest yellow instances and the closest purple instances from the solid line is the same.

The maximization of this distance is the optimization target of SVMs: try to find a line (or rather a hyperplane) that is as far away as possible from the actual data. The rational for this is roughly this:

- Our training data is only a sample.
- It stands to reason that there are more data points, especially also more extreme data points, that are not part of the sample.
- This means that these data points could fall into the region between the dashed lines, though it is likely that they are close to the dashed lines.
- If we were to use one of the dashed lines as decision boundary, we would misclassify these extreme data points.
- By maximizing the margin and using the solid line, we minimize the likelihood of misclassification due to extreme data points.

The SVM gets their name from the *support vectors*. These are the circled instances on the dashed lines. These instances all have the same distance to the decision boundary and are responsible for the margin. Training with only the support vectors would lead to the same result as training with all data, because there would still not be a better line to separate the data.

However, SVMs would not be as popular as they are, if they could only learn the linear decision boundaries. The SVMs use the *feature expansion* to learn non-linear decision boundaries. The idea is to calculate new features and then find a linear decision boundary in this transformed feature space with a higher dimension. The linear decisions look non-linear in the lower dimensional space. The following figure visualizes this concept.

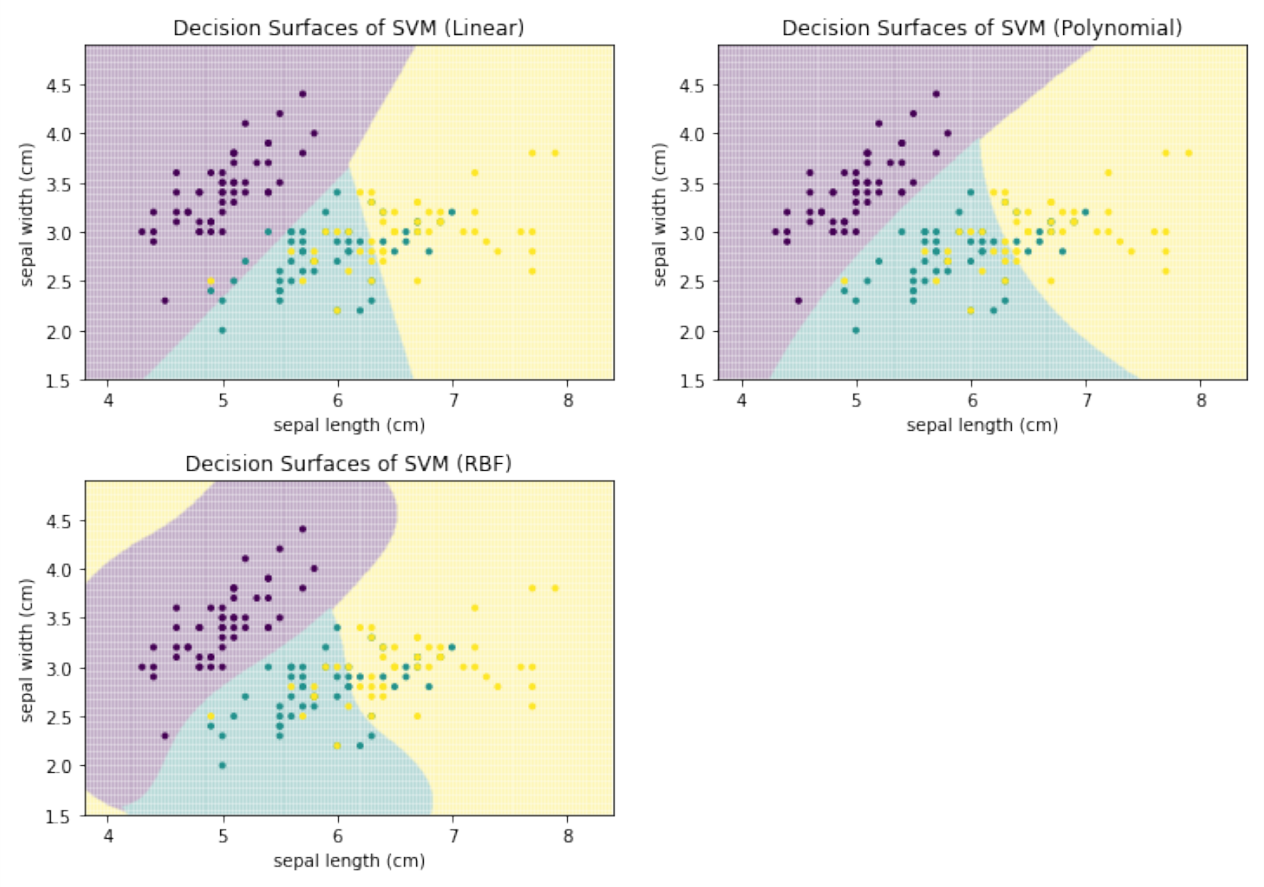


On the left side, we see two circles as sample data. It is obvious that there is no line that can separate the purple instances from the yellow instances. Thus, a linear separation with an SVM is not possible. The right side shows the expansion of the data with a new feature ($x_1^2 + x_2^2$). When we look at the data in three dimensions, we see that the yellow data is at the bottom and the purple data is at the top. We can just cut this data with a plane to separate the data linearly, which was not possible with the original dimensions.

The feature expansion is done using *kernel functions*. The kernel function defines how new features are calculated. The above example used a *polynomial* kernel, where the feature space is expanded using a quadratic function. This transformation can still be visualized. Other kernel transformations are more complex, e.g., the *Radial Basis Function* (RBF). Here, the radial distances between the instances are the foundation of the kernel and the resulting feature space has infinite dimensions. This can neither be visualized, nor easily used for calculations. Thus, it is - in general - not

possible to visualize how the expanded feature look like. Internally, SVMs avoid to actually expand the features using the *kernel trick*, which we do not discuss in greater detail.

The shape of the decision surface depends on the kernel that is used for the feature expansion. The visualization below shows how the decision boundaries for the Iris data look like for a linear SVM (i.e., no kernel), a polynomial SVM, and a SVM with an RBF kernel.



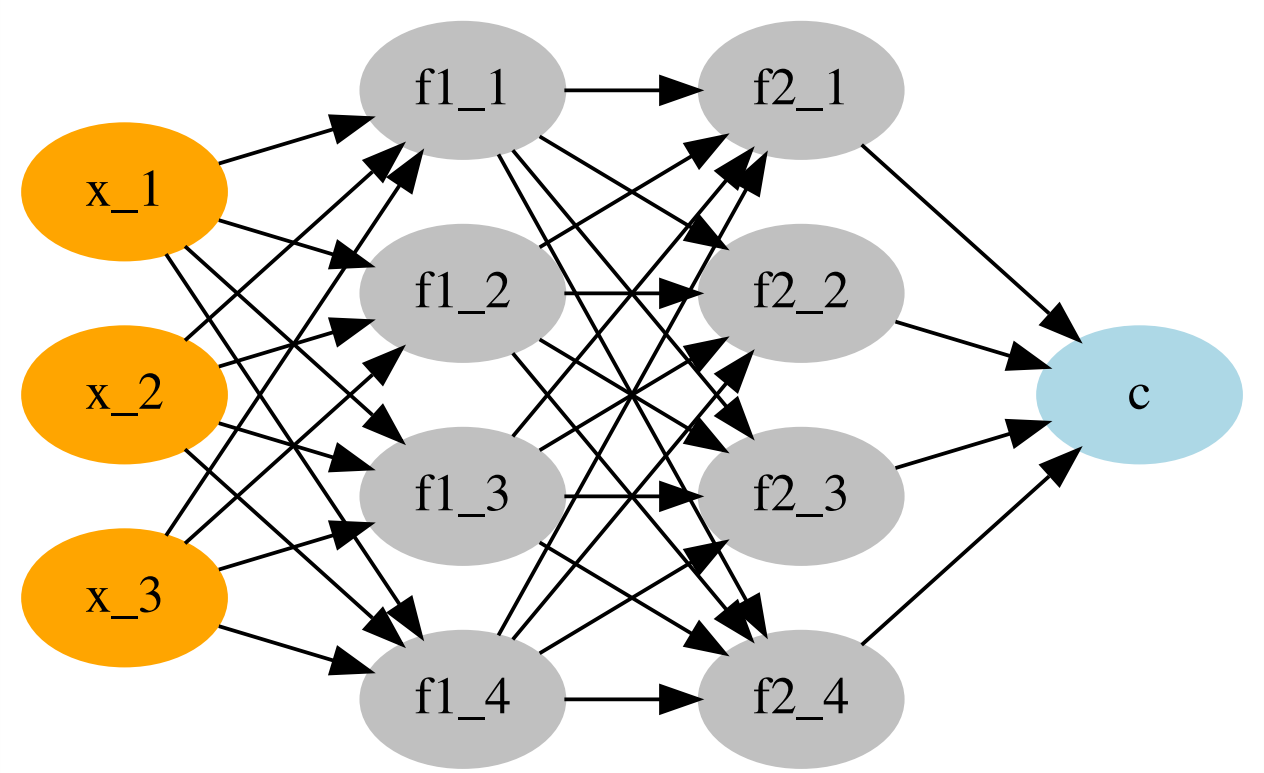
Note:

Kernel functions and SVMs have additional parameters. When you want to use SVMs in practice, you should learn more about these parameters, as they can make a big difference in the classification performance. SVMs can be very powerful, but this depends a lot on the selection of a good kernel function with appropriate parameters.

7.10 Neural Networks

Neural networks are based on the idea to mimic the communication between neurons in the human brain to make decisions. The general idea is relatively simple. Neurons are *activated* and *propagate* the values of their activation to other neurons. These neurons then determine their activation based on the propagated input from other neurons.

A simple neural network may look like this.

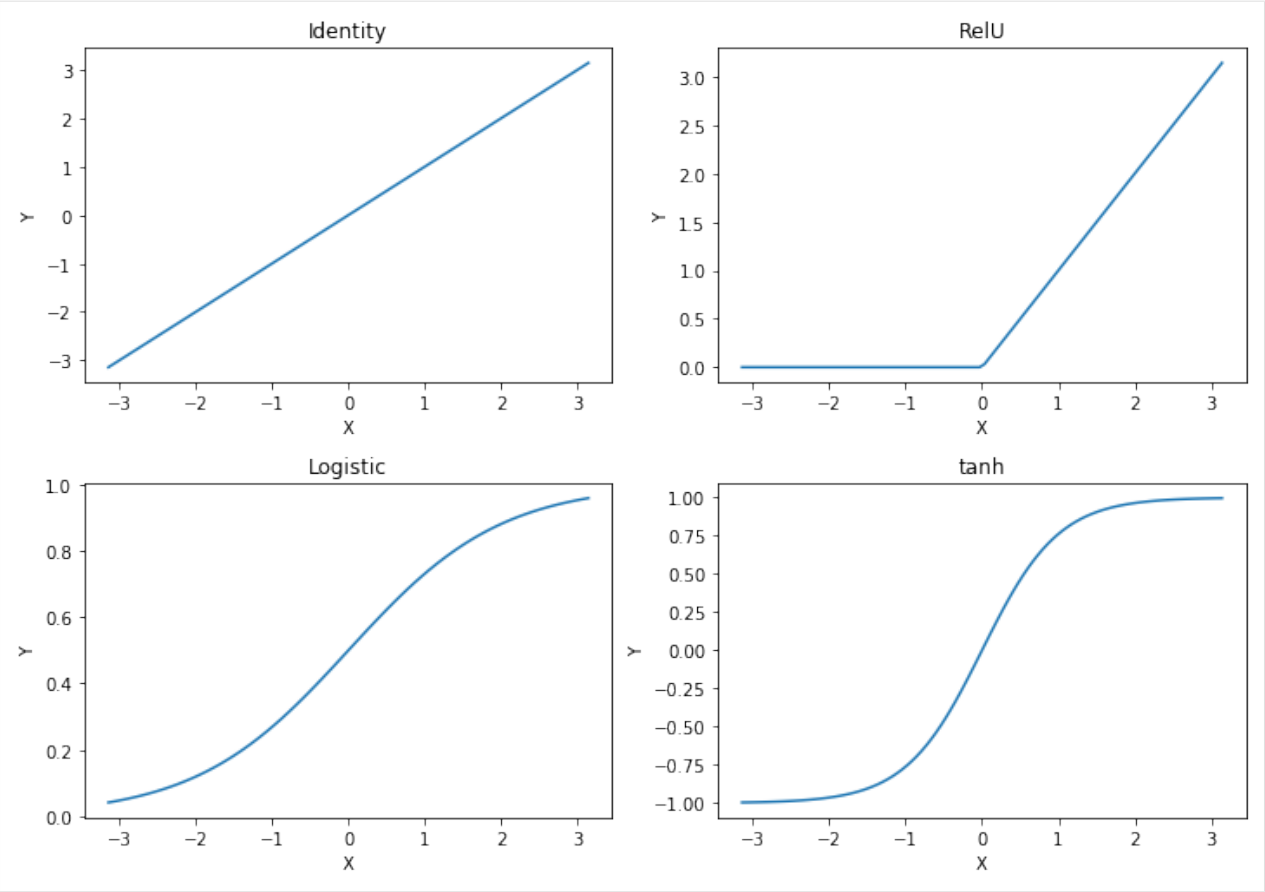


Each node in the network represents a neuron. The neurons are organized in layers. We differentiate between the *input layer* (orange), the *hidden layers* (gray), and the *output layer* (blue). The input layer has one neuron for each feature and each of these neurons represents one feature. The example has three such neurons for features x_1, x_2, and x_3. The output layer contains our value of interest, in our case, the predicted class c. The hidden layers contain additional neurons and are used to correlate the information from the input layer in order to determine the value of the neurons in the output layer. These layers are *hidden*, because users of neural networks do not observe what happens in these layers directly, but only indirectly through the input and the output. The edges between the neuron define to which other neurons the activation of the neurons is passed. In the example above, the we show *fully connected layers*, i.e., the activation is forwarded to each neuron in the next layer. A network that consists only of fully connected layers is also known as *Multilayer Perceptron* (MLP). There are many different kinds of neural network structures. However, a discussion of the different kinds of networks and their use cases is out of scope.

Each neuron uses the weighted sum of the inputs to calculate its activation function. For example, the value of $f_{1,3}$ is calculated as

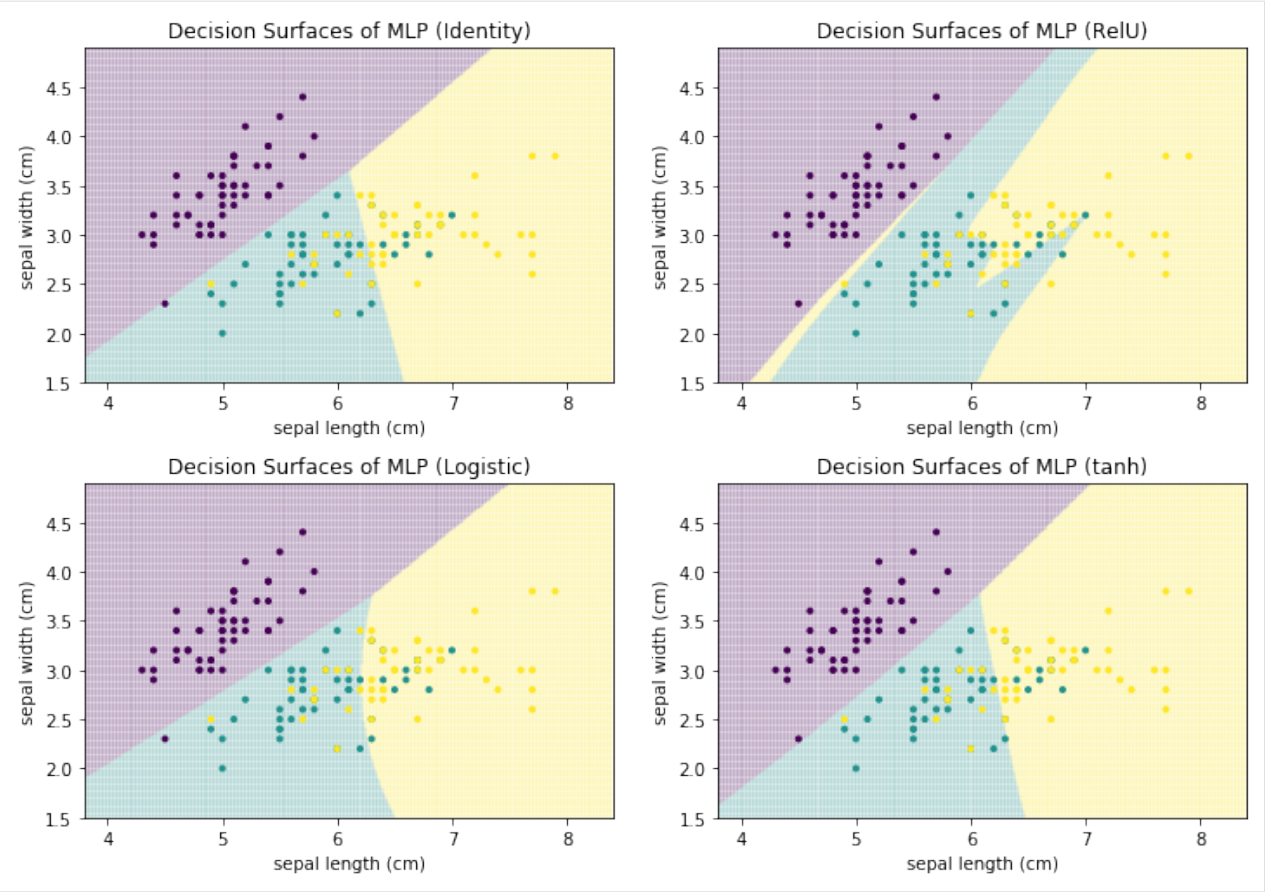
$$f_{1,3} = f_{act}(\sum_{i=1}^{3} w_i x_i)$$

where f_{act} is the *activation function* of the neuron and w_i are the *weights* of each input. The activation function further transforms the input. Different activation functions have different effects. The visualization below shows four common activation functions.



The simplest activation function is the identity, i.e., the weighted inputs are the activation of the neuron. The *rectified linear unit*, commonly just referred to as *RelU* is a variant of the identity, where all negative values are set to 0, i.e., there is no negative activation. The Logistic function and the tanh both have "S shapes". We already know the logistic function from the logistic regression. The logistic function maps the input to a probability distribution: if the weighted input is negative, the probability is less than 0.5, if the weighted input is positive, the weighted input is greater than 0.5. At around -3/+3, the output converges to 0/+1. *tanh* is roughly linear, if the weighted input is between -1 and 1. For values outside this range, the output of tanh quickly converges to -1/+1. Thus, with the exception of the identity, the other activation functions somewhat "normalize" extreme inputs.

The user of the neural network has to define the complete structure of the neural network, i.e., how many hidden layers there are, how many neurons there are on each layer, how the neurons are connected with each other to propagate their activations, and the activation function. The training of the neural network is the estimation of the weights, i.e., the influence of each input neuron on the activation function of a neuron. Below, we show how the choice of the activation function affects the training of a MLP for the Iris data, with three fully connected hidden layers that have 100 neurons each.



As we can see, the decision surfaces all look different. Identity has linear decision boundary, Logistic and tanh both have a small curve in the decision boundaries and the shape of RelU can only be described as irregular. RelU also shows us the big risk when using neural networks: overfitting. Three fully connected layers with 100 neurons on each layer already have $2 * (100 * 100) = 20000$ weights that must be estimated, without even accounting for the weights between the input layer and the first hidden layer, and the last hidden layer and the output layer. This huge number of parameters that must be estimated allowed the MLP with the RelU function to carve out a small yellow area for a single instance. This is a general problem of neural networks: if they are too small, they do not allow for good performance, if they are too large, there will be overfitting. Therefore, any kind of neural network should be used with care and the parameters should be carefully selected, e.g., with the help of a validation set, to prevent overfitting.

Note:

This is not a "deep" dive into neural networks, we only scratch the surface of a vast amount of options you have with neural networks. There are different layer structures, e.g., convolution, pooling, or dropout layers. Networks can also have "backward edges", leading to *recurrent* neural networks, or even have special kinds of neurons like Long-Short-Term-Memory (LSTM) networks. In general, there are different *network architectures*, where different architectures are well suited for different purposes, e.g., convolutional neural networks for object recognition, LSTM networks for text mining, or generative adversarial networks (GANs) to improve training or create artificial data. Thus, before using deep learning, there is a vast amount of options that you must be aware of, so you can define appropriate network structures. Please be aware that powerful neural networks usually require large amounts of data to achieve a good performance. Thus, if you only have a limited amount of data available, other options are often better than neural networks.

7.11 Comparison of Classification Models

The seven different approaches for classification all have different strengths and weakness. Each of the algorithms may be the best choice, depending on the circumstances. Within this section, we directly compare the algorithms to each other.

7.11.1 General Approach

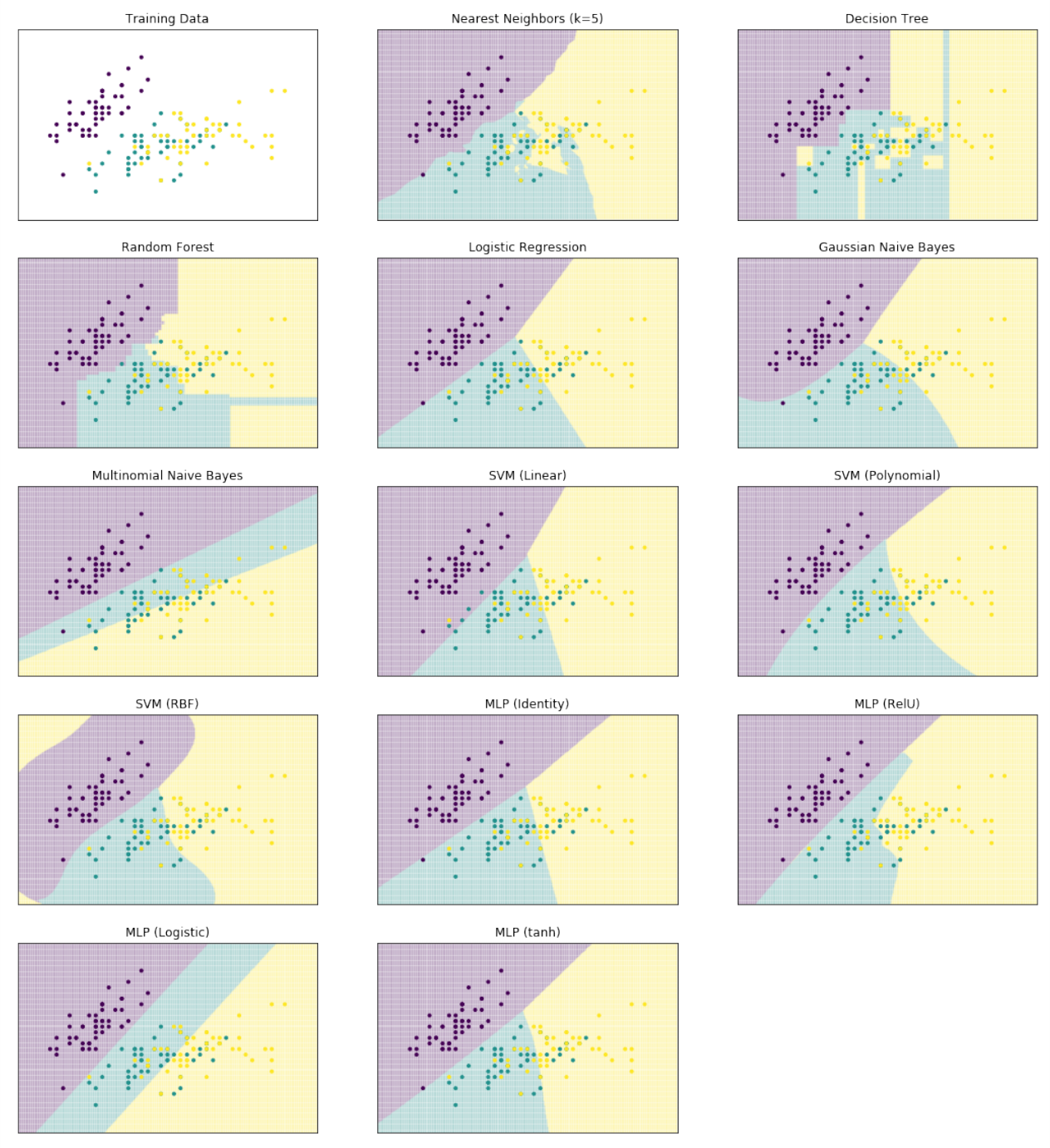
We selected the algorithms, because they all represent a different approach towards building a classification model:

- k-Nearest Neighbor is instance-based.
- Decision Trees are rule-based and use concepts from information theory.
- Random Forests are an example for ensemble learners.
- Logistic Regression shows how regression can be used for classification.
- Naive Bayes is grounded in the principles of probability theory.
- SVM use the concept of margins and feature expansion through kernels.
- Neural Networks are a very generic model that can be tailored to specific purposes. However, underneath everything, they are just very complex regression models.

This selection is by no means complete, for example, we did not consider an ensemble learner that uses *boosting*. However, the selection of algorithms gives a good overview of how different algorithms work and what their strengths and weaknesses are.

7.11.2 Decision Surfaces

We start by comparing the decision surfaces on different data sets. We summarize the results for the Iris data that we used so far below.

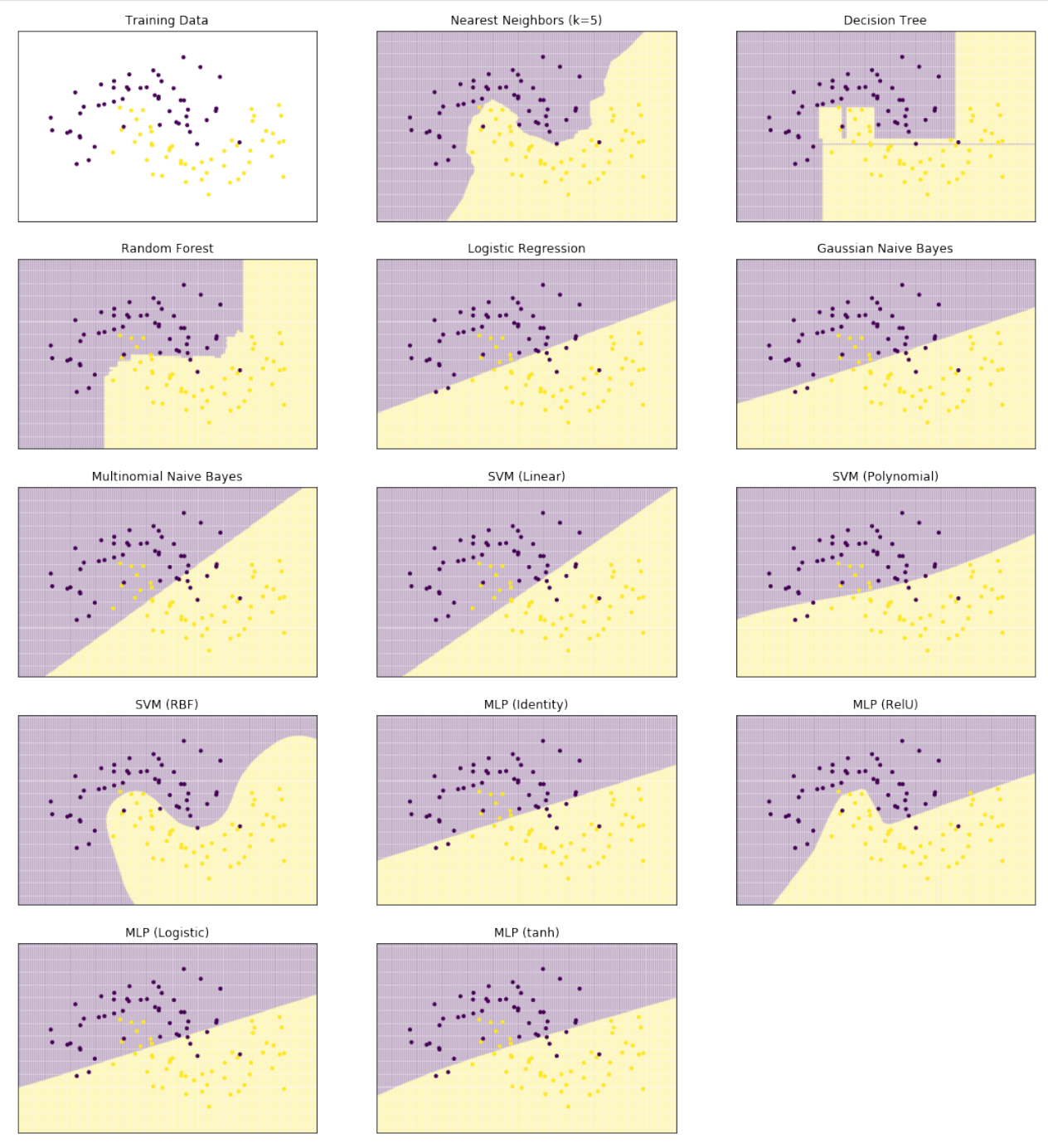


We first note that all classifiers are good at separating purple from the others. However, there are major differences regarding the separation of teal and yellow. The k-Nearest Neighbor and Decision Tree all have islands of teal within the yellow area, which may be overfitting. This is possible, because these classifiers work instance-based, respectively using a set of rules that can describe disjoint regions. This would also be easily possible with Random Forests, but is avoided due to the small trees that are used for the ensemble. The decision boundaries of all other classifiers are described by continuous functions, which usually do not allow such islands. The only really bad classifier is the Multinomial Naive Bayes, which is to be expected (also for the other examples), because we consider numeric data, for which Multinomial Naive Bayes is not well suited.

We can observe that while the decision surfaces are very similar in the regions where training data is located, there

are big differences, in regions further away from the training data. This is a general property that you should be aware with classification: it is unclear how well the classifiers generalize beyond the distribution of the training data. Thus, classification models can only be used reliable, if the training data has the same distribution as the test data.

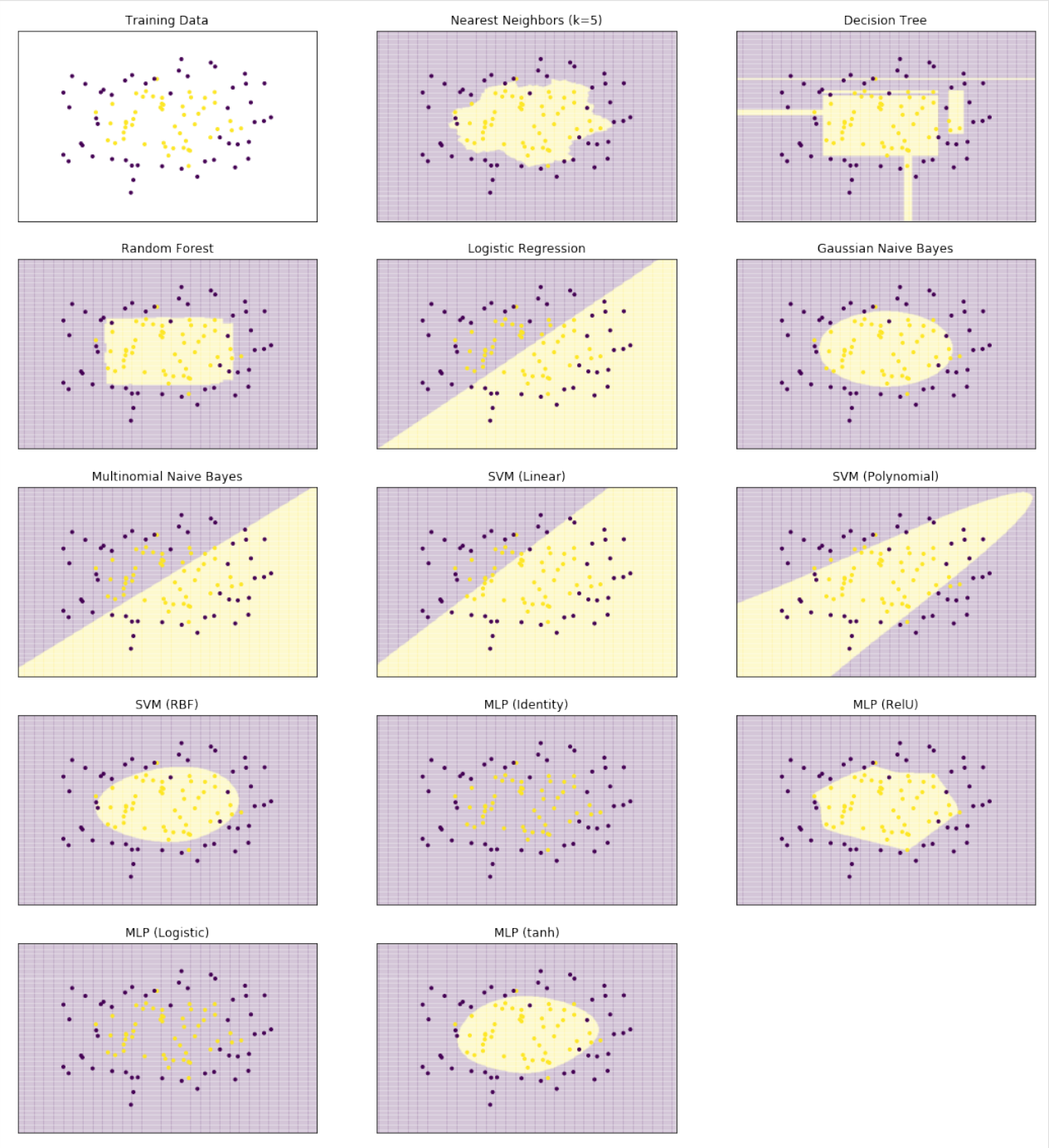
The next data set we consider are two half-moons that are very close to each other, but that have (almost) no overlap.



With this data, we start to clearly see the limitations of classifiers. Logistic regression, Naive Bayes, the linear and polynomial SVM, as well as all MLPs except RelU fail to separate the data and instead have linear (or almost linear) decision boundary with misclassification in the regions where the moons overlap. While these models are not extremely bad, because most data is classified correctly, they fail to derive decision boundary that match the actual distribution of the classes. Other algorithms are better able to differentiate between the two half-moons and the deci-

sion boundary matches the actual shape of two classes. We see a slight tendency for overfitting again with the Decision Tree.

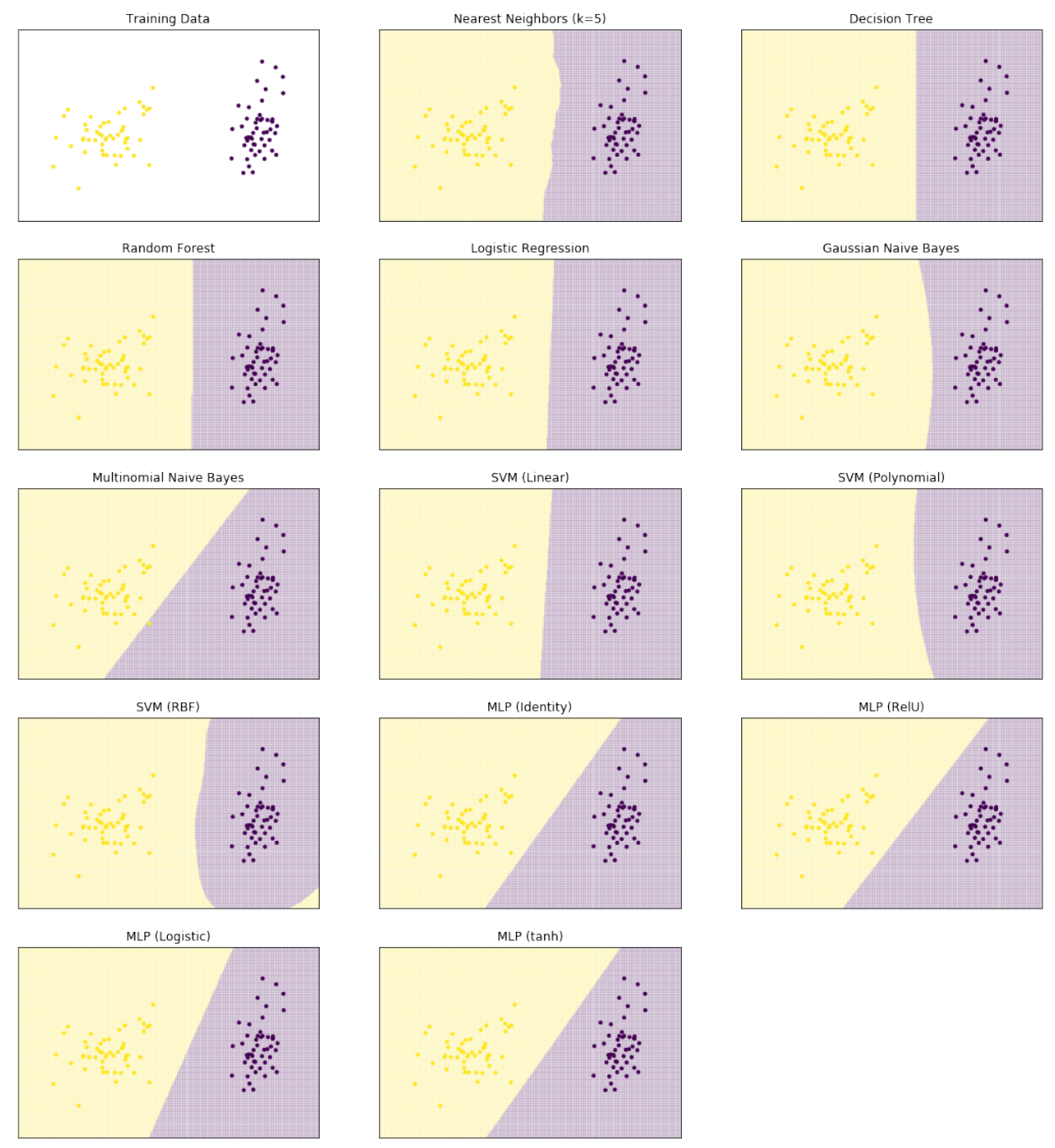
The next data set is a circle within another circle. Again, the two data sets are very close to each other, but only very few instances actually overlap.



The results are, in general, similar to the half-moons: the algorithms that worked for the half-moons also work well for the circles. However, there are also some algorithms that failed on the half-moons but perform well with the circles, i.e., the Gaussian Naive Bayes, and the MLP with tanh as activation function. This can be explained by the fact that the almost linear separation worked reasonably well with the half-moons. Thus, there was not so much pressure to use a different, more complex decision surface. This is not the case with the circles, that only work with round decision

surfaces. This put pressure on the internal optimization of the algorithms and forced them to determine the better and more complex decision boundaries.

Finally, we consider clearly separated data with a large gap.

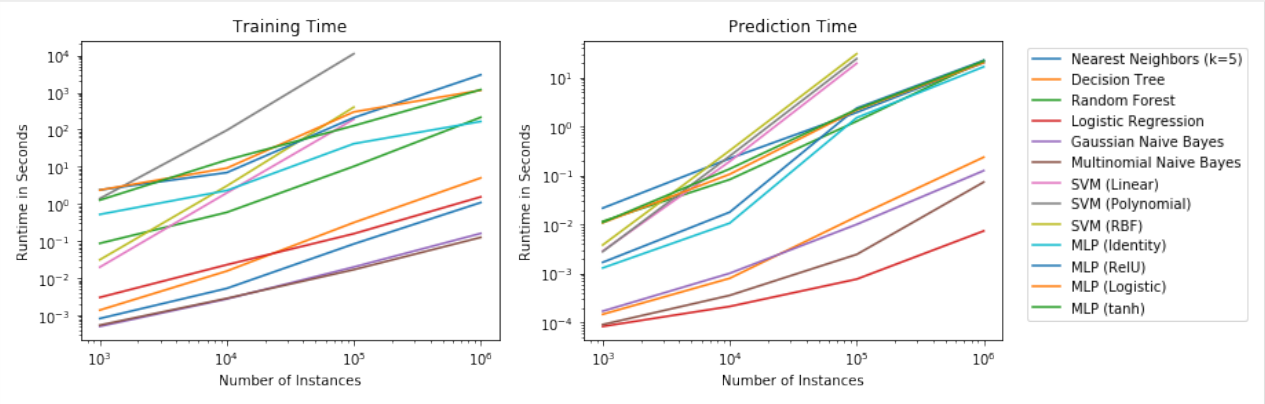


As expected, all algorithms can easily predict this data correctly. However, we observe differences in the margins, that we already know from the SVM. We see that most algorithms estimate the decision boundary such that the margin is large. However, the MLP and also the Multinomial Naive Bayes have decision boundaries with small margins. For the Multinomial Naive Bayes, this is likely because of the numeric data and not a general failing of the algorithm. For the MLP, this is because the optimization criterion completely ignores the margin. As a result, the MLP will use the first separating line that is found, regardless of the margin. Since no other line would be an improvement in terms of

misclassifications, they are not considered improvements and the optimization stops.

7.11.3 Execution Time

The execution time can also be an important factor, especially if the training or prediction must be performed under strict time constraints or if there is a large amount of data. Below, we measured the execution times on the half-moon data for 1,000 to 1,000,000 instances. The measurements are done with a normal laptop (Intel Core i7-8850 @ 2.60GHz, 32 GB RAM) with scikit-learn 0.23.1.



First, we observe that the increase in training time is roughly linear for all models. However, the slope for the SVMs is steeper than that of all other models. Due to that, we did not measure the execution time for 1,000,000 instances. Other than that, it is clearly visible that the training for some models is fast. Decision Trees, Logistic Regression, Nearest Neighbors, and Naive Bayes require roughly the same amount of time for 1,000,000 instances for training, as a MLP for 1,000 instance. The Random Forest is somewhere between the MLP and the other faster models. We note that the training time of the Random Forest depends on the number of trees that is used and increases linearly with the number of trees. In the example above, we used 100 trees. If we would have used 1,000 trees, the execution time would be similar to the MLP.

The results for the prediction time are roughly similar to the results of the training time, regarding the differences between the models. The outlier is Nearest Neighbors. This is because there is no real training, and many pairwise distance must be calculated for the predictions. We also note that the differences in execution time between the MLP and the Random Forest vanish for the predictions.

Please note that execution times can vary depending on the data and parameters. We only compared how the execution times change if we increase the number of instances. The number of features can also have a strong effect on the execution times. Moreover, especially for neural networks, the architecture of the neural network is important for the execution time. Training of a huge neural network with millions of neurons on millions of instances usually requires specialized hardware.

7.11.4 Explanatory Value and Concise Representation

With the exception of the k-Nearest Neighbor algorithm, that requires the complete training data, all other models have a concise representation that can be serialized and used elsewhere. However, only the representations of the Decision Tree and Logistic Regression can directly be read and used by humans.

There are big differences in the explanatory value of these representations. The Decision Trees can be interpreted by almost everyone, a computer science or data science background is not required. Consequently, Decision Trees can also be discussed with domain experts. The Decision Tree indirectly also shows the importance of the features, as more important features are higher in the tree. While Random Forest usually provides better predictions, they have less explanatory value than decision trees. That a human can read hundreds of decision trees and understand how a decision is made through their majority is unrealistic. However, what can be gained from Random Forests is the

importance of the features: the more often a feature is used within the trees, the more important the feature. This analysis is supported by so-called *feature importance maps*.

Logistic regression has a high explanatory value, because the coefficients can be interpreted as odds ratio. Therefore, logistic regression is well suited to understand the impact of the features on the result. However, this requires more training and effort than for decision trees.

The remaining models have almost no reliable explanatory value. For Naive Bayes, the individual probabilities may be considered, but they are usually not reliable because of the naive assumption of independence. The interpretation of neural networks is currently a very active field of research, e.g., by visualizing the weights of the network or the activation of the neurons, to better understand what influences the decisions.

7.11.5 Scoring

For $k > 1$, the average of the neighbors may be used as score for the k-Nearest Neighbor algorithm. However, these scores are usually not very good. If scores are relevant, a different algorithm should be used. The counts of the different classes in the partitions of a Decision Tree can be used to determine the scores. Through averaging, this can be generalized to Random Forests. Logistic Regression directly estimates the probability of the outcome and, hence, the score. The scores of these three algorithms are *well calibrated*, which means that they can be used as reliable estimates of the probability that the prediction is correct. Naive Bayes also provides scores that can be used. However, the scores of Naive Bayes are usually not well calibrated, because of the naive assumption. SVMs do not support scoring at all. A trick to allow some scoring with SVMs is to create a separate regression model to estimate the scores for the predictions of a SVM. However, if scores are important, a different algorithm that natively supports scoring should be used. Neural networks also support scoring. If the scores are well calibrated depends on the neural network structure and the target function of the training.

7.11.6 Categorical Features

Only the Decision Tree, Random Forest, and Multinomial Naive Bayes work without restrictions for categorical features. For all other algorithms, one-hot encoding can be used to support categorical features. This usually works well, except for the k-Nearest Neighbor algorithm. The k-Nearest Neighbor algorithm relies on distance measures, which are not useful for one-hot encoded data.

7.11.7 Missing Features

None of the algorithms can easily deal with missing features. Decision Trees and Random Forests may work to some degree with random features. For example, a feature may not be in all subtrees of a Decision Tree. All subtrees without this feature can still be used, which means many instances may still be predicted correctly. For Random Forest, one could just ignore all trees in the ensemble, in which the feature is present. Especially if the number of trees in the ensemble is huge, this approach should work well. For all other algorithms, the computations do not make sense with missing features.

While k-Nearest Neighbor cannot work with missing features, this algorithm is commonly used to *impute* the values for missing features (see, e.g., Barnard and Meng, 1999). Hence, one may argue that missing features are actually a strength of k-Nearest Neighbor.

7.11.8 Correlated Features

Most of the algorithms can deal with correlated features. Decision Trees select features by their importance. If there are correlated features, this means that these features will just be less important, as soon as the first of the correlated features is part of the tree. The same is, in general, true for Random Forests. However, if there are many correlated features in comparison to uncorrelated features with actual information, this may mean that the uncorrelated features are only part of few trees in the ensemble and, thereby, degrade the prediction performance. For neural networks, correlations are not a problem, because the weights can account for the correlations. Similarly, the prediction performance of Logistic Regression is also not negatively impacted by correlated features. However, the explanatory value of Logistic Regression may suffer, because coefficients of correlated features may not be reliable (see Chapter 8).

For Naive Bayes, correlated features are breaking the naive assumption of independence. Thus, the more correlations there are, the stronger the violation of the independence assumption and the results are less reliable. However, in practice the impact of this is usually just that the scores are a very bad estimator for the probability, the prediction performance is often not negatively affected. For SVMs, the impact of correlated features depends on the kernel. A linear SVM is not impacted by correlations. However, if a kernel function that uses distances between instances is used, e.g., a RBF kernel, the correlated features are overrepresented in the distance, which may affect the prediction performance. Due to this problem, k-Nearest Neighbor is also negatively affected by correlated features.

7.11.9 Summary

The following two tables summarizes the different strengths and weaknesses of the classification algorithms.

	Decision Surface	Runtime	Explanatory Value	Concise Representation
k-Nearest Neighbor	+	o	−	−
Decision Tree	o	+	+	+
Random Forest	+	o	−	o
Logistic Regression	−	+	+	+
Naive Bayes	−	+	−	o
SVM	+	−	−	o
Neural Network	+	−	−	o

	Scoring	Categorical Features	Missing Features	Correlated Features
k-Nearest Neighbor	−	−	+	−
Decision Tree	+	+	o	+
Random Forest	+	+	o	+
Logistic Regression	+	o	−	o
Naive Bayes	+	+	−	−
SVM	−	o	−	−
Neural Network	+	o	−	+

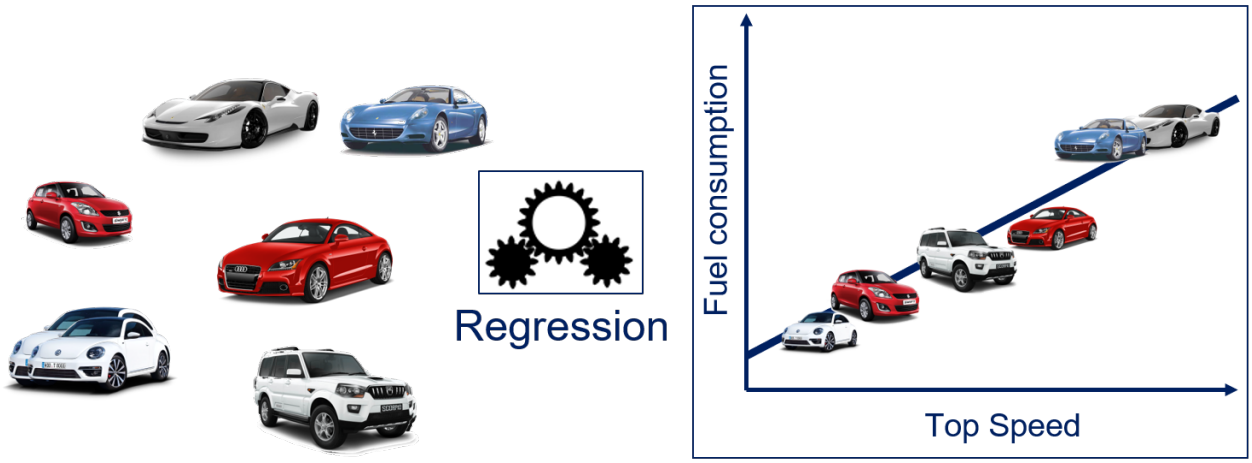
CHAPTER

EIGHT

REGRESSION

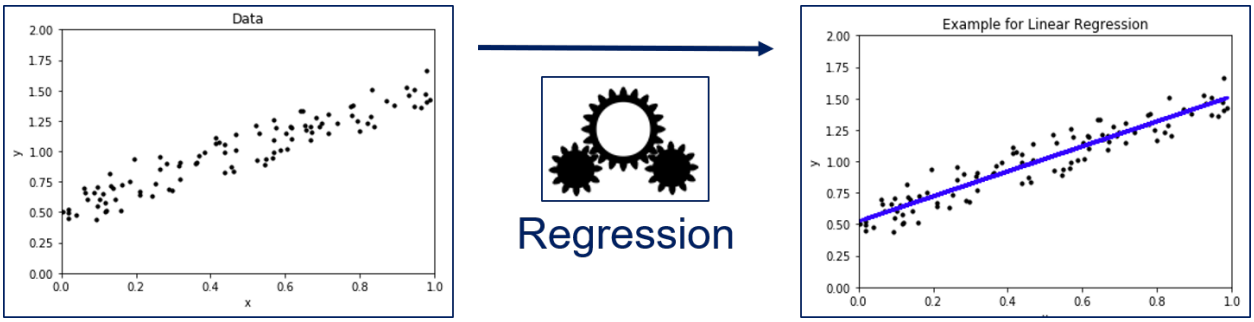
8.1 Overview

Regression is about finding the relationship between variables. Consider the following example.



We may be interested in the relationship between different properties of cars. Two important properties are fuel consumption and the top speed of cars. Usually, cars that can reach a higher top speed have stronger motors that consume more fuel. With regression, we can model the relationship between the variables using data about a sample of cars.

A bit more abstract, classification looks as follows.



We have a set of instances for which features are known. In regression analysis, the features are also often called the *independent variables*. We want to know the relationship between the features and a *dependent variable*, i.e., our value of interest. In the figure above, x is our feature and y is our dependent variable. Through the regression model we can describe how y changes, when x changes. The example shows a *Linear Regression*, i.e., regression through a linear model, which is a line in two dimensions.

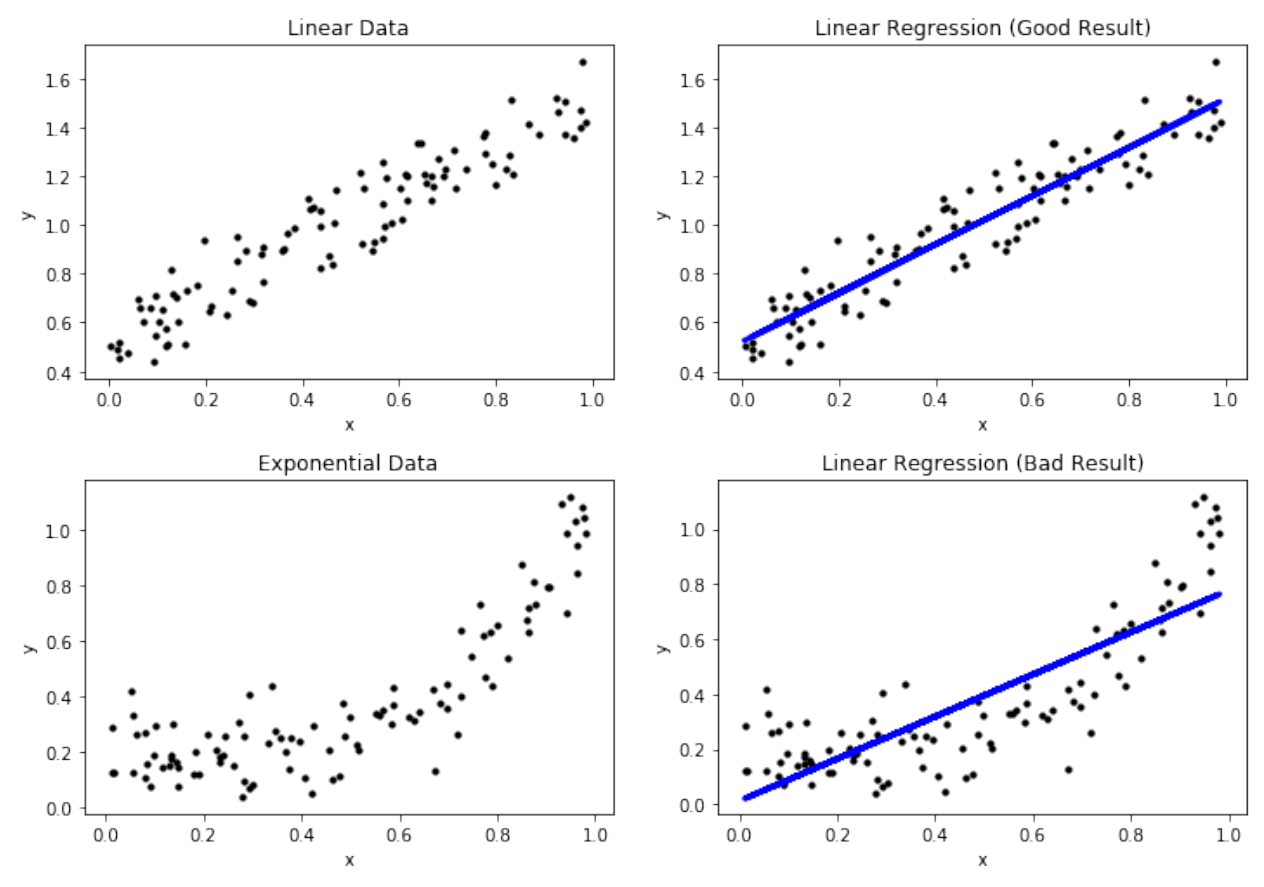
8.1.1 The Formal Problem

Formally, we have a set of objects $O = \{object_1, object_2, ...\}$ that may be infinite. Moreover, we have representations of these objects in a feature space $\mathcal{F}\{\phi(o) : o \in O\}$ that we assume is a subset of the real values, i.e., $\mathcal{F} \subseteq \mathbb{R}^m$. For each object, our value of interest is a dependent variable $f^*(o) = y \in \mathbb{R}$.

The task of regression is to estimate a regression function $f : \mathcal{F} \to \mathbb{R}$ such that $f(\phi(o)) \approx f^*(o)$.

8.2 Performance of Regressions

Same as with classification, an important question is how we can estimate if a regression function is a good approximation of the dependent variable. We discuss the performance of Linear Regression using the two examples below.



The first example shows a very good regression model, where the regression function depicted in blue matches the data. This is expected, because we generated the data with a linear expression and use a linear expression model. The second example shows a bad regression model, because the regression does not really match the data. This is expected, because we fit a linear model to exponential data. In the following, we refer to the first example as the good result, and the second example as the bad result. In terms of regression, good/bad results are also called a good/bad *fit* of the data

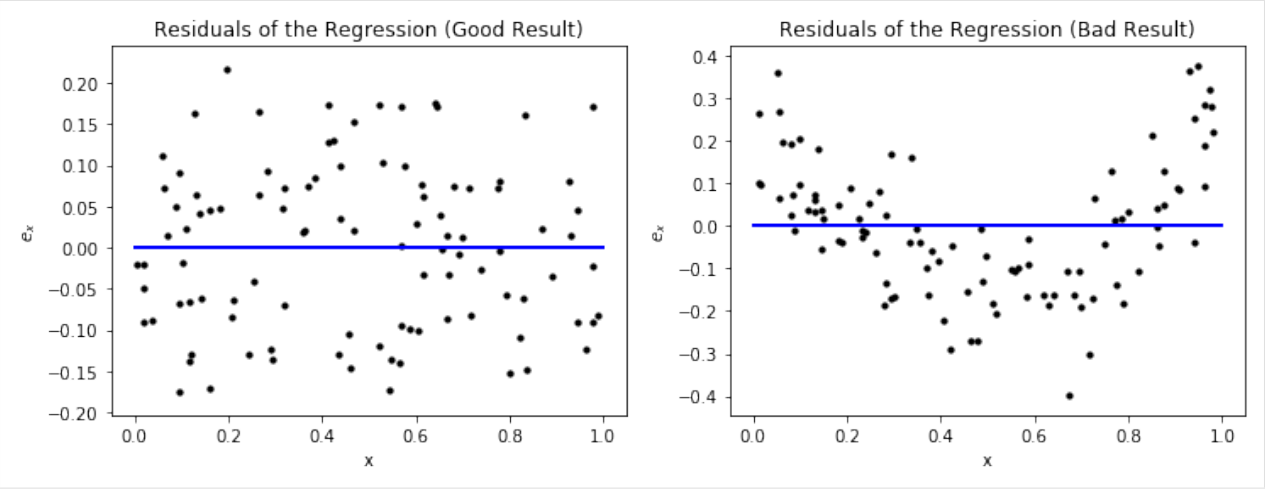
8.2.1 Visual Performance Assessment

The plots above already show the first method for the assessment of the quality of a regression model. We can plot the actual data and the regression function together in the same plot. Through this, we can see if the regression is close to the actual data and, therefore, likely a good fit. Moreover, we can see systematic errors in the regression. In the good result, we see that the actual data is evenly scattered around the regression line without a large deviation. Hence, we can see that there are small errors in the regression model, but there is no clearly discernable pattern in these errors. This is different for the bad result: at the beginning, the instances are all above the blue line, between 0.4 and 0.7 the instances are below the blue line and at the end the instances are, again, all above the blue line. This clear pattern is a strong indicator for a bad fit, because there is clearly an aspect of the data that is not covered by the regression model. When the instances are all above the blue regression line, we say that the regression function *under predicts*, when the instances are all below the regression line, we say that the regression function *over predicts*.

Another way to look at the results of a prediction model visually is through the *residuals*. The residual of an instance $x \in \mathcal{F}$ for a regression function f is defined as

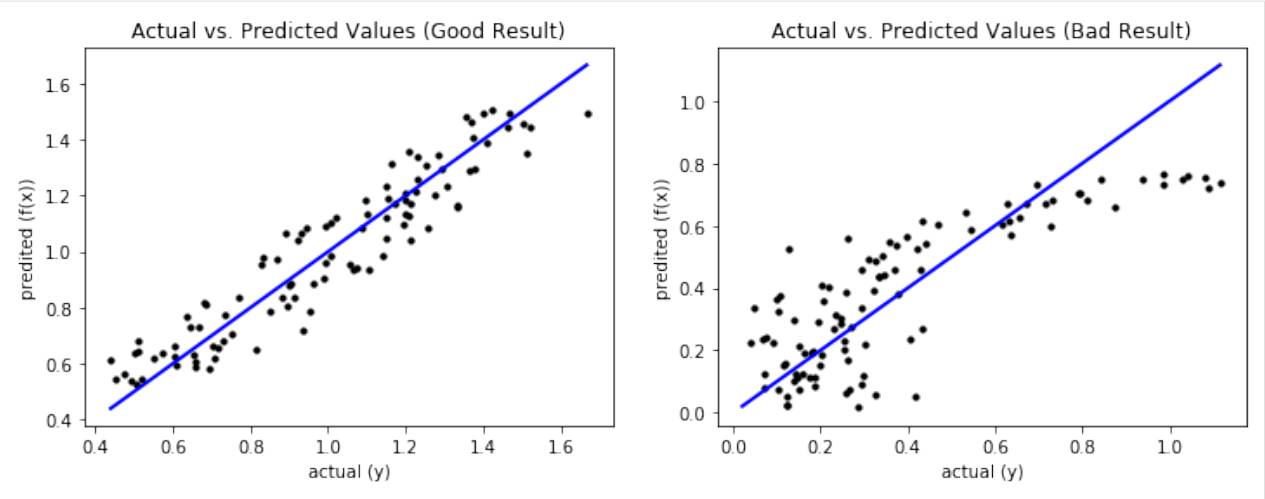
$$e_x = f^*(x) - f(x) = y - f(x),$$

i.e., the difference between the actual value and the prediction. We can plot the residuals versus the values of a feature, to see how the deviation evolves.



The residuals provide a similar view on the error of the regression function as we have seen above. However, just looking at the plots, one may think that the left plot for the good result is actually bad, because the spread from the blue line, which indicates correct predictions, looks large. However, a closer look at the y-axis reveals that this is only because the absolute values of the results are very small. If the y-axis would have a larger range, a bad result could indeed look similar. Thus, it is always important to consider the scale of the y-axis of residual plots in order to investigate the absolute magnitude of the residuals. For the bad result, we see the same over and under prediction patterns we observed before. We also see that the range of the y-axis is much larger than for the good result, which also indicates that this is not a good result.

The drawback of the above visualizations is that they only work for single features. However, there is another way of visual analysis that allows us to gain insights into the data. The idea is similar to the confusion matrix we know from Chapter 7: we plot the actual values of the dependent variable versus the predicted values.



Ideally, the prediction equals the actual values, which is on the diagonal of the plot. The further away the data is from the diagonal, the worse the prediction. For a good result, the predictions are close to the diagonal and the deviations are random without a clear pattern, similar to our prior visual analysis. For a bad result, the deviations from the diagonal are larger and there may be patterns that show systematic errors. This is the case for the bad result. For the data where the actual values are less than 0.35, the prediction is completely random, which we can see due to the random pattern with large deviations from the diagonal in this area. For larger actual values, we see systematic errors: first there is over prediction, then there is under prediction.

8.2.2 Performance Metrics

There are also several performance metrics that can be used to assess the quality of regression models. These metrics are based on the residuals. Let $X = \{x_1, ..., x_n\} \subseteq \mathcal{F} \subseteq \mathbb{R}^m$ a sample of instance with m features and $Y = \{y_1, ..., y_n\} \subseteq \mathbb{R}$ the corresponding dependent variables.

Metric	Description	Definition
Mean Absolute Error	The arithmetic mean of the absolute residuals, i.e., the mean absolute deviation of the predictions from the actual values.	$MAE = \frac{1}{n}\sum_{i=1}^{n} \lvert e_{x_i} \rvert$
Mean Squared Error	The arithmetic mean of the squared residuals.	$MSE = \frac{1}{n}\sum_{i=1}^{n} e_{x_i}^2$
R Squared, R2, R^2	The coefficient of determination, which is defined as the fraction of the variance of the data that is explained by the regression model.	$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - f(x_i))^2}{\sum_{i=1}^{n}(y_i - mean(Y))^2}$
Adjusted R Squared	Adjustment of R^2 that takes the model complexity into account	$\bar{R}^2 = 1 - (1 - R^2)\frac{n-1}{n-m-1}$

The MAE measures the absolute difference between the predictions and the actual values. The similar measure MSE uses the squared distances instead of the actual distances. Because the MSE uses the squared residuals, it is more sensitive to outliers in the residuals than MAE. Thus, MSE increases strongly with outliers, while these may be hidden with MAE. The biggest problem with MAE and MSE is that both are absolute measures for the error and, thus, hard to interpret because there is no fixed reference point. The performance measures for classification were mostly distributed in the interval [0,1] and it was clear what a perfect result looks like and what the worst possible result looks like. To interpret MAE and MSE, detailed knowledge about the dependent variable is required, especially about the scale of the dependent variable and the meaning of differences.

In that respect, R^2 is more similar to the metrics we know from classification and is distributed in the interval [0,1], where 1 is a perfect value. R^2 answers the question how much better the regression function is than just using the

arithmetic mean of the data. The sum in the denominator is over the residuals of the arithmetic mean as the regression function. In the nominator, we have the sum of the residuals of the regression. If we would divide both the nominator and denominator by $|X|$, we would have the MSE as nominator and the variance as denominator. In other words, R^2 measures the ratio of the MSE of the model and the variance of the dependent variable. The lower the MSE in relation to the variance, the better.

A weakness of R^2 is that this usually increases with more features, which can lead to overfitting. Adjusted R^2 addresses this weakness by taking the number of features into account. Thus, if the decrease in MSE is less than the penalty of adjusted R^2 for the number of features, adjusted R^2 increases.

We now look at the values of these performance metrics for our example.

```
Performance Metrics (Good Result)
MAE:         0.08
MSE:         0.01
R2:          0.89
Adjusted R2: 0.89

Performance Metrics (Bad Result)
MAE:         0.13
MSE:         0.03
R2:          0.68
Adjusted R2: 0.68
```

We directly observe the problem with MAE and MSE: because we have randomly generated data, MAE and MSE have no real meaning for us. We only see that the values of the bad result are larger than the values for the good result: about 60% higher for MAE and about 3 times higher for MSE. Please note that the values for MSE are less than the values for MAE, because $x^2 < x$ for $x \in (0, 1)$. This further emphasizes that the interpretation of MAE and MSE is difficult.

For R^2 we also see that the value of the good result is much larger than of the bad result. The bad result explains only about 2/3 of the variance, while the good result explains almost 90%. Adjusted R^2 is not different, because we only have a single feature.

8.3 Linear Regression

We already encountered Linear Regression, e.g., in Chapter 7 when we discussed Logistic Regression or in the example above. We now take a closer look at how Linear Regression works. The formula of the regression function of Linear Regression is

$$y = b_0 + b_1 x_1 + ... + b_m x_m = b_0 + \sum_{i=1}^{m} b_i x_i$$

with *coefficients* $b_1, ..., b_m \in \mathbb{R}$ and the *intercept* b_0. The coefficients define how a feature influences the dependent variable. The coefficients are closely related to the correlation between the dependent variable and the features. Usually, a positive coefficient means that the feature is positively correlated to the dependent variable, a negative coefficient means that there is a negative correlation. However, there are also exceptions to this general rule, as we will see in the next sections.

The intercept is the "base value" of the dependent variable, i.e., the value of the dependent variable if all features would be zero. This concept can be visualized as the location where a regression line *intercepts* the axis of the coordinate system.

The key aspect of Linear Regression is how the coefficients and intercept are determined, as this can affect not only the quality of the regression function, but also how the coefficients may be interpreted.

8.3.1 Ordinary Least Squares

The standard way to estimate the coefficients of Linear Regression is using the *Ordinary Least Squares* (OLS) method. For this, we interpret the instance as matrix and the dependent variables and coefficients as vectors:

$$X = \begin{pmatrix} x_{1,1} & \dots & x_{1,m} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \dots & x_{n,m} \end{pmatrix}$$
$$y = (y_1, \dots, y_n)$$
$$b = (b_1, \dots, b_m)$$

Because of this, we can use matrix multiplication to calculate all the predictions given a set of coefficients as a single vector, i.e.,

$$f(X) = b_0 + Xb.$$

Since we want to minimize the residuals, we can formulate this as an optimization problem

$$\min ||f(X) - f^*(X)||_2^2 = \min ||b_0 + Xb - y||_2^2.$$

Hence, we minimize the square of the Euclidean norm (see Chapter 6), in other words we are looking for the solution with the *least squares*.

Because we have more instances than features (otherwise we have too many features), there are usually multiple optimal solutions for this optimization problem. While these solutions are the same with respect to the OLS, there can be interesting effects if the features are correlated with each other. This can be demonstrated with an example.

Consider two features x_1, x_2 and a dependent variable y such that $y = x_1 = x_2$. Here are some examples for optimal solutions:

- Solution 1: $b_0 = 0, b_1 = 0.5, b_2 = 0.5$
- Solution 2: $b_0 = 0, b_1 = 1, b_2 = 0$
- Solution 3: $b_0 = 0, b_1 = 0, b_2 = 1$
- Solution 4: $b_0 = 0, b_1 = 10000, b_2 = -9999$

In general, any solution with $b_0 = 0$ and $b_1 + b_2 = 1$ is optimal. The first three solutions we listed seem reasonable. However, a solution with very large coefficients is also possible and optimal. This also demonstrates that the coefficients of features can be misleading in terms of correlation with the dependent variable. In the example above b_1 and b_2 can assume any real value and, thus indicate any possible correlation.

8.3.2 Ridge

In the following, we show how OLS can be modified with *regularization* to find the first three solutions and avoid the problems with correlated features. The idea of regularization is simple: when the optimization problem allows multiple solutions, some of which are more desirable than others, we modify the problem such that favorable solutions are better. This is done by adding a *regularization term* to the OLS formula.

Ridge regularization is based on the idea that smaller values for coefficients are better than larger values. Thus, we should select the optimal solution that minimizes the values of the coefficients. To achieve this, the OLS problem is modified such that

$$\min ||b_0 + Xb - y||_2^2 + \alpha ||b||_2^2$$

where $\alpha \in \mathbb{R}, \alpha > 0$ regulates the strength of the regularization term. To demonstrate the effect of this regularization, we calculate $||b||_2^2$ for the four solutions of our example above:

- Solution 1: $||b||_2^2 = \sqrt{0.5^2 + 0.5^2}^2 = 0.5$
- Solution 2/3: $||b||_2^2 = \sqrt{1^2 + 0^2}^2 = 1.0$
- Solution 4: $||b||_2^2 = \sqrt{10000^2 + (-9999)^2}^2 = 199980001$

Thus, ridge would select the first solution.

8.3.3 Lasso

Another commonly used regularization technique is *lasso*. While ridge leads to small coefficients, the correlated variables are still both non-zero. A model with similar performance, that uses fewer features is often better, because the likelihood of overfitting is reduced. The goal of lasso is to not only minimize the coefficient values of correlated variables, but to achieve that they become exactly zero. For this, the OLS problem is modified such that

$$\min ||b_0 + Xb - y||_2^2 + \alpha||b||_1,$$

i.e., the difference to ridge is the norm that is used. Ridge uses the squared euclidean distance, whereas Lasso uses the Manhattan distance. When we look at the solutions, we get the following.

- Solution 1: $||b||_1 = |0.5| + |0.5| = 1.0$
- Solution 2/3: $||b||_1 = |1| + |0| = 1.0$
- Solution 4: $||b||_1 = |10000| + |-9999| = 19999.0$

Thus, Solutions 1, 2, and 3 would be optimal for lasso, which means it is likely that a solution is selected, where a coefficient is exactly zero.

> Note:
>
> We do not provide a detailed explanation of why Ridge only minimizes the values of the coefficients, while Lasso tries to find coefficients that are exactly zero, here. The short explanation is that the regularization term can also be considered as a constraint on the optimization problem. The shape of this constraint is a circle for Ridge and a diamond for Lasso, because that is the shape of the unit circles of the Euclidean and the Manhattan norm. Due to the diamond shape, values of exactly zero are likely with Lasso. A full explanation for this, can, e.g., be found here.

8.3.4 Elastic Net

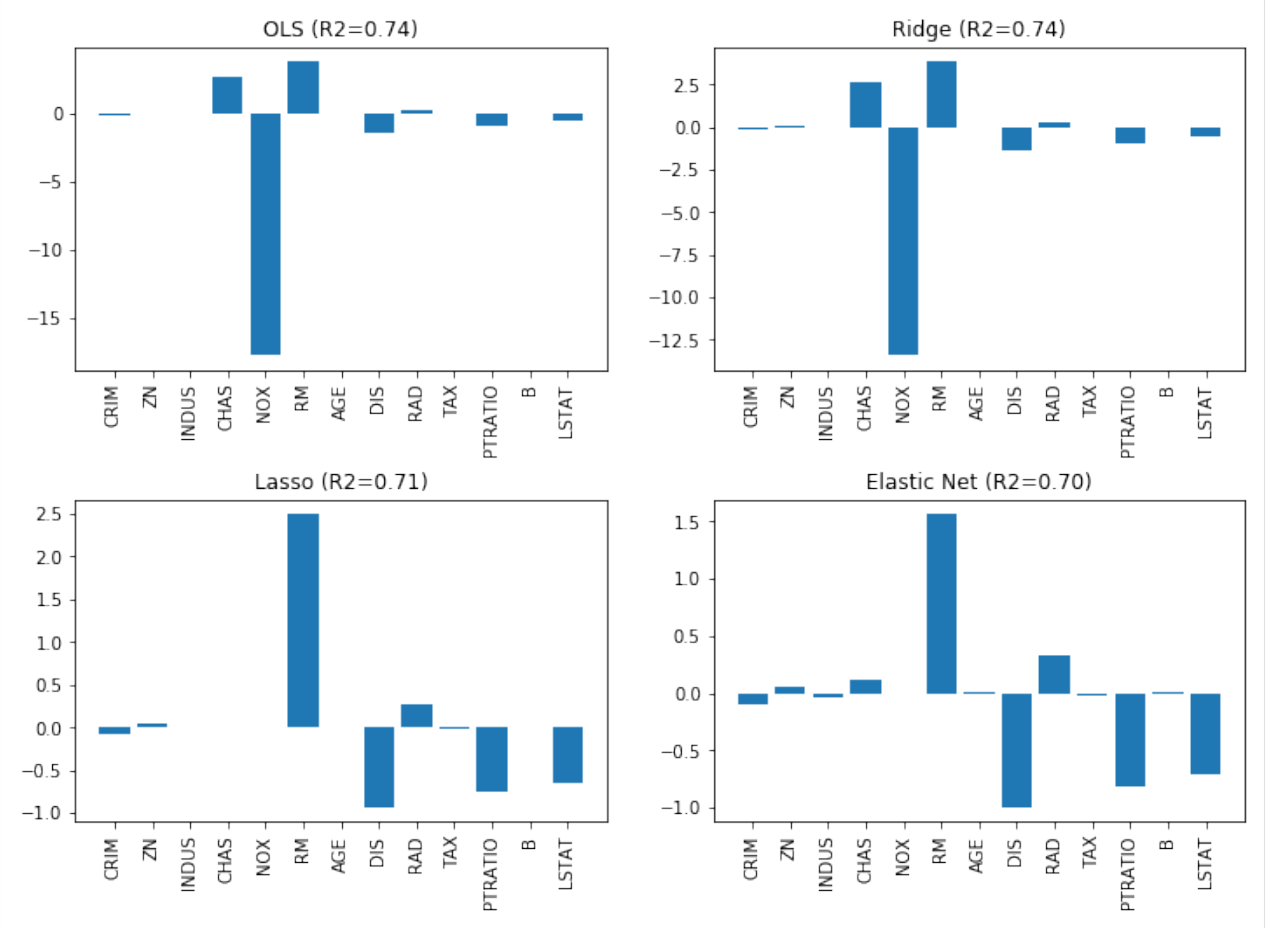
The last regularization variant is the *elastic net*. The elastic net combines ridge and lasso regularization and uses both the squared Euclidean norm and the Manhattan norm, i.e.,

$$\min ||b_0 + Xb - y||_2^2 + \rho\alpha||b||_1 + \frac{1-\rho}{2}\alpha||b||_2^2$$

where $\rho \in [0, 1]$ defines the weighting between ridge and lasso. With $\rho = 0$, the elastic net uses only ridge regularization, with $\rho = 1$, the elastic net uses only lasso regularization. The goal of combining lasso and ridge is to enforce low coefficients, but ideally also try to reduce them to exactly zero, if possible.

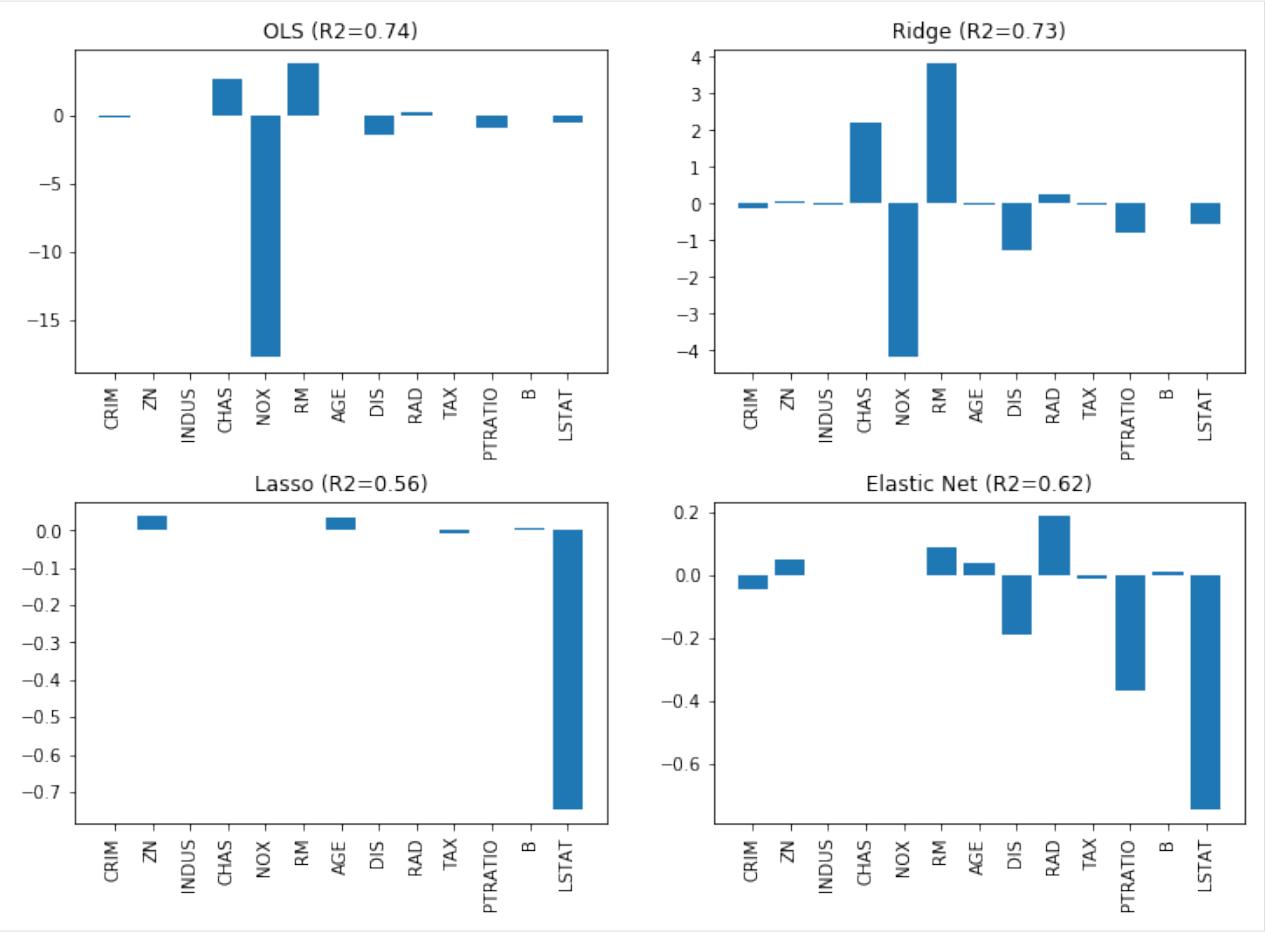
8.3.5 Impact of Regularization

We now revisit the Boston data, we already know from Chapter 3 and create a Linear Regression for the data. We compare the results of OLS, Ridge, Lasso, and the Elastic Net. We use $\alpha = 0.5$ as regularization strength and $\rho = 0.25$ as ratio between Ridge and Lasso for the Elastic Net. The bar charts show how the coefficients change.



We can see the effects quite clearly. The value of the largest coefficient, NOX is the most interesting value. We clearly see how this coefficient changes for the different models: with OLS the value is around 18, with Ridge it drops to around 13, and with Lasso to exactly zero and is eliminated. With Ridge, there is not even an impact on the R^2 score, which remains unchanged. Lasso drops not only NOX, but also INDUS, CHAS, and AGE. Even though fewer features are used, the R^2 score drops only slightly. With the Elastic Net, fewer features are dropped, but NOX is still dropped and the overall value of the coefficients are lower than with Lasso.

Overall, we see a strong effect of the regularization on the coefficients and also observe that regularization must be used with care: if the regularization is too strong, the focus of the optimization problem is not to find a good fit, but to find low coefficients. Consider the same example, but with $\alpha = 5$, i.e., ten times stronger regularization.



While Ridge still works and the coefficients are further reduced, Lasso and Elastic Net focus too much on the coefficients the R^2 value drops drastically. Thus, the choice of α should always be evaluated with respect to the goodness of fit with OLS.

8.4 Beyond Linear Regression

Linear Regression is obviously not the only way to build regression models and not sufficient for complex problems. The regression function does not have to be linear, but can also be, e.g., polynomial. Polynomial regression should be used with great care, because it tends to overfit, especially for higher degree polynomials. Moreover, with the exception of Naive Bayes, all algorithms discussed in Chapter 7 have a regression variant: k-Nearest Neighbor Regression, Regression Trees, Random Forest Regression, Support Vector Regression, and Neural Network Regression. The general idea for all these approaches remains the same, only details are modified to better fit the regression context. For example, Regression Trees usually use the variance to determine the most informative features.

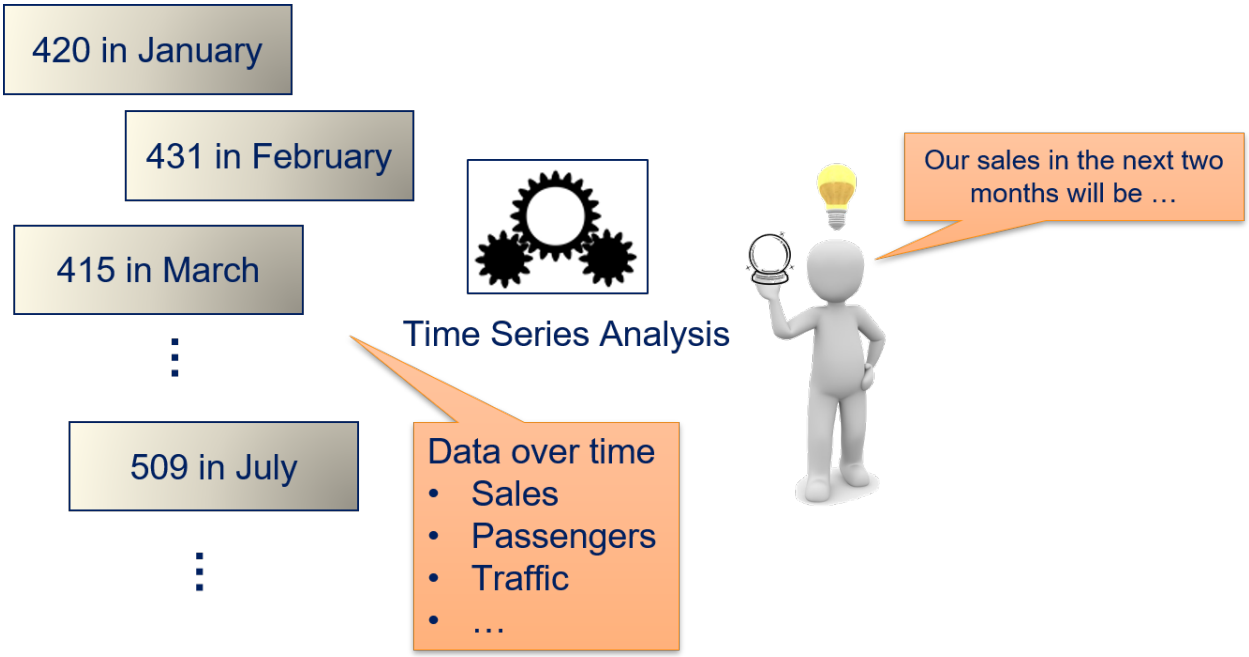
CHAPTER

NINE

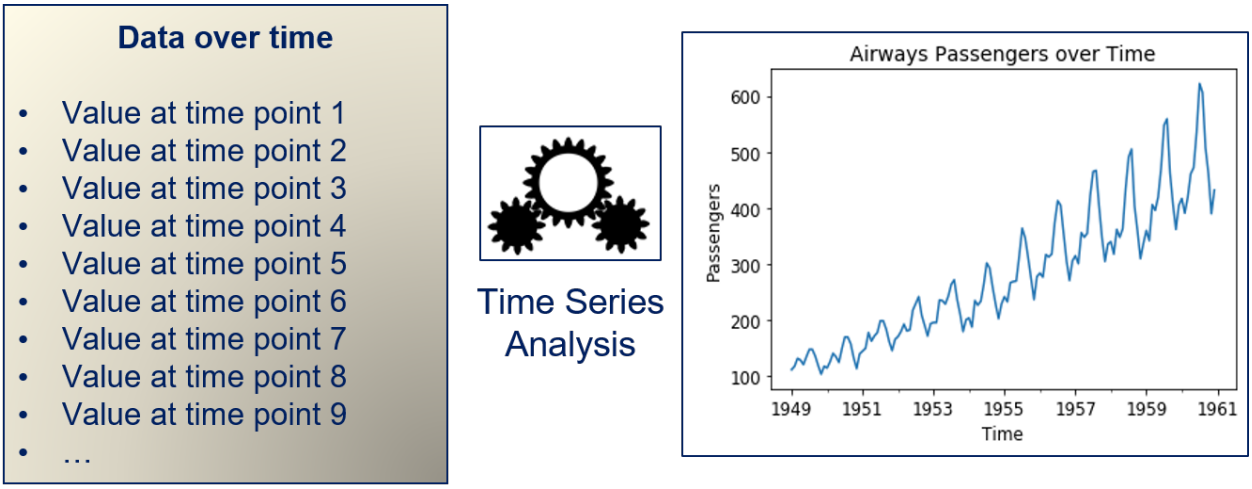
TIME SERIES ANALYSIS

9.1 Overview

Time series analysis deals with ordered sequences of data, e.g., data over time. Consider the following example.



The data we use for the time series analysis are passenger numbers measured every month. Thus, we have an instance for every month in the data. The goal of the time series analysis is to describe the temporal structure of the data, i.e., to find a model that explains how the data changes from month to month. This could be done, e.g., to predict the next value of a time series. A bit more abstract, time series analysis can be described as follows.



Our data are values at enumerated points in time. The time series model describes how the data changes over time. This problem is related to regression: if we were to say that the time is our feature and the value at that time is our dependent variable, we would have a very simple time series analysis.

9.1.1 The Formal Problem

Formally, we have discrete values $\{x_1, .., x_T\} = \{x_t\}_{t=1}^T$ with $x_t \in \mathbb{R}$. We use the notation $\{x_t\}$ as shorthand for $\{x_t\}_{t=1}^T$. Mathematically, these values can be seen as a series of random variables or as a stochastic process. A core assumption of the time series analysis is that the difference between the time t and the time $t+1$ is equal for all $t = 1, ..., T-1$. Thus, we have a fixed step size for the time, e.g., minutes, hours, days, weeks, months, or years.

A time series consists of different *components*. We consider three such components:

- The *trend* of the time series T_t. The trend is the overall change over time, e.g., a steady growth or decline in values. An example for the trend is, e.g., the constantly growing energy consumption in most countries.
- The *seasonality* of the time series S_t. The seasonality is the result of *seasonal effects*, i.e., regularly recurring effects that lead to an increase or decrease with respect to the general trend. An example for a seasonal effect is, e.g., the change in energy consumption over the year, because it is longer dark outside.
- The *autocorrelation* between observations R_t. The autocorrelation models how the value at time t depends on the prior values, i.e., how x_t is correlated with $x_{t-i}, x_{t-2}, ...$ This autocorrelation models the changes in the time series that are not explained by the trend or seasonality.

These three components together define the value of our time series, i.e.,

$$x_t = T_t + S_t + R_t.$$

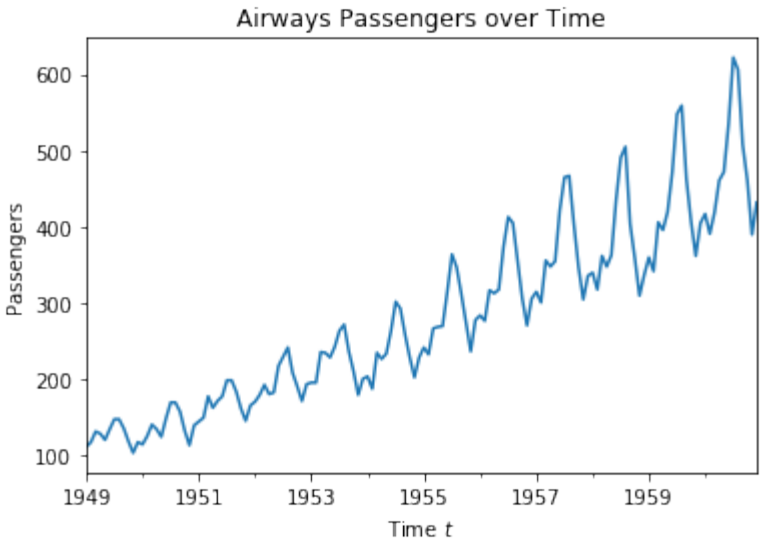
9.2 Box-Jenkins

We consider a special case of time series models, so called *Box-Jenkins* models. With the Box-Jenkins approach, we first need to ensure that the time series is *stationary* and can then model the autocorrelation as a stochastic process that combines the dependencies on past values with a random component.

Stationary means that the mean value and the variance are constant and do not change over time. For time series, this means that we must first determine the trend and seasonality of the data. The trend is a model for how the mean value of a time series changes over time, the seasonality models how the mean value of a time series changes during seasons. If both are removed, the remaining autocorrelation should be stationary. Because the autocorrelation is modeled for stationary data, Box-Jenkins models achieve this with only few parameters.

In the following, we consider each aspect of a Box-Jenkins approach for time series analysis in detail. First, we discuss how the trend and seasonality can be determined and removed from the data. Afterwards, we look at ARMA models for the autocorrelation.

We explain the time series analysis using a running example, i.e., data about air passengers from 1949 to 1961. The data is measured monthly, i.e., the difference between two points is one month.



9.3 Trend and Seasonal Effects

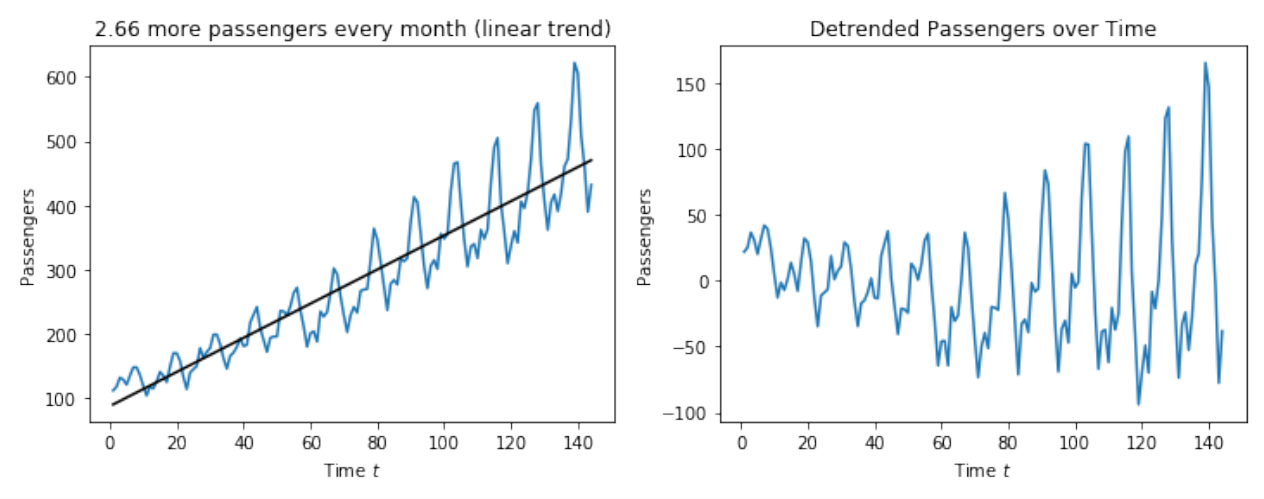
We make the time series stationary by modeling the trend term T_t and the seasonality S_t. The general workflow is always the same:

- We model the trend T_t of the time series $\{x_t\}$.
- We calculate the *detrended* time series $\hat{x}_t = x_t - T_t$.
- We model the seasonality S_t of the detrended time series $\{\hat{x}_t\}$.
- We calculate the *detrended and seasonally adjusted* time series $\hat{\hat{x}}_t = \hat{x}_t - S_t$

We consider two different approaches for modeling the trend and seasonality.

9.3.1 Regression and Seasonal Means

The first approach towards detrending is to use a regression model. The trend of the air passenger data looks linear, i.e., like a steady increase of customers over time. In this case, we can fit a linear regression model to explain the trend.



The plot on the left shows how the linear regression fits the time series. We see that the fit is good and matches the shape of the time series. The slope of the linear regression, i.e., the coefficient is the monthly increase of passengers. Thus, this model for the trend gives us insights into the trend itself: there are 2.66 more passengers every month.

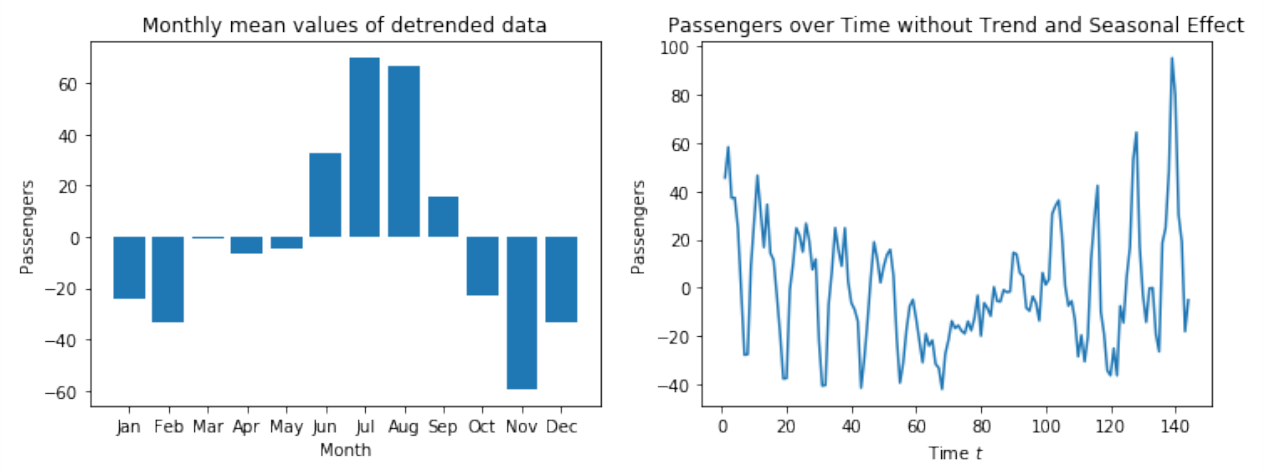
The plot on the right shows the time series without the trend. We see that there is no increasing trend anymore, the values seem to be centered around zero. This means that the trend was successfully removed. We also observe that the range of the values on the y-axis decreased from $600 - 100 = 500$ to $150 - (-100) = 250$. This means that the trend explains about fifty percent of magnitude the change in the values over time.

However, we still observe that there is a regular pattern in the fluctuation around zero. This is the seasonality of the data. Recall that the data we use are monthly measurements. Therefore, it is a reasonable conjecture that the seasonality that we observe is a recurring pattern in the same month every year. We can further explore this by adjusting the seasonality through the *seasonal mean*. To calculate the seasonal mean, we must know the length of a season, i.e., after how many time steps the seasonal pattern repeats itself. In case of the sample data, we have a *monthly mean* with a length of $s = 12$ timesteps for the seasons. We can then calculate the seasonal pattern as the difference of the overall mean of the detrended time series and the seasonal mean and substract this from the detrended time series, i.e.,

$$\hat{\hat{x}}_t = \hat{x}_t - (mean(\{\hat{x}_{t'}\}_{\{t' \in 1,\dots,T : mod(t',s) = mod(t,s)\}} - mean(\{\hat{x}_t\}))$$

where $mod(t, s)$ is the modulo of t divided by s. For example, if we have $s = 12$ and $t = 26$, $mod(t, s) = 2$. In that case, the seasonal mean would be calculated over all values t' where $mod(t', s) = 2$, i.e., 2, 14, 26, 38, etc.

For our sample data with a seasonality of $s = 12$ we get the following.



We see that there are over 60 passengers more in July than on average, in November there are about 60 passengers less than average. The seasonal effect of the other months fluctuates between these values. March has almost no seasonal effect. Thus, we again get a great explanatory value for the seasonality, same as for the trend.

When we use these monthly mean values for seasonal adjustment, we get the time series we see on the right. The range of the y-axis decreased to only $100 - (-40) = 140$. Thus, the seasonality and the trend explain most of the variation in the data. However, the plot shows that there is still a regular pattern. Please note that every original pattern had a single upwards spike and two downwards spikes next to each other. After seasonal adjustment, we have:

- Two upwards spikes and one downwards spike before $t = 70$.
- Almost no upwards/downwards spikes between $t = 70$ and $t = 100$.
- One upwards spike and two downwards spikes after $t = 100$.

This shows that the seasonality is not fully corrected. The reason is, that in this data, there is a trend in the seasonal effect: same as the overall number of passengers, the monthly fluctuations also get stronger over the years. In the middle of the data, the seasonal mean matches well with the data, hence no more spikes. At the beginning of the data, the seasonal mean is larger than the actual seasonal effect, which means the seasonal adjustment is too strong and the seasonal pattern is inverted. At the end of the data, the seasonal mean is less than the actual seasonal effect, which means that there is still something of seasonal effect left.

Overall, our time series is still not fully stationary, as there are still seasonal patterns in the data. However, since the pattern is now relatively weak (decrease in range of the y-axis), it would be possible to use this data to model the autoregressive part of the time series.

9.3.2 Differencing

The second approach for modeling the trend and seasonality is based on *differencing*. Differencing is similar to the derivative of a function and more powerful than the adjustment through regression and seasonal means. The idea behind differencing is that the trend is nothing more than the slope of the time series. The slope is nothing more than the first derivative of the time series. If the trend of the time series is not constant, but changes over time, this would be the second order derivative. Thus, if we want to remove the trend, we can use the derivative to do this.

Differencing us usually implemented with a variant of the *point slope equation* to estimate the slope of a line. The slope between two points $(x_1, y_1), (x_2, y_2)$ is defined as $\frac{y_2 - y_1}{x_2 - x_1}$. We use that we have equidistant time steps t, i.e., to points next to each other in the time series are $(t-1, x_{t-1}), (t, x_t)$ and the slope is

$$\frac{x_t - x_{t-1}}{t - (t-1)} = \frac{x_t - x_{t-1}}{1} = x_t - x_{t-1}.$$

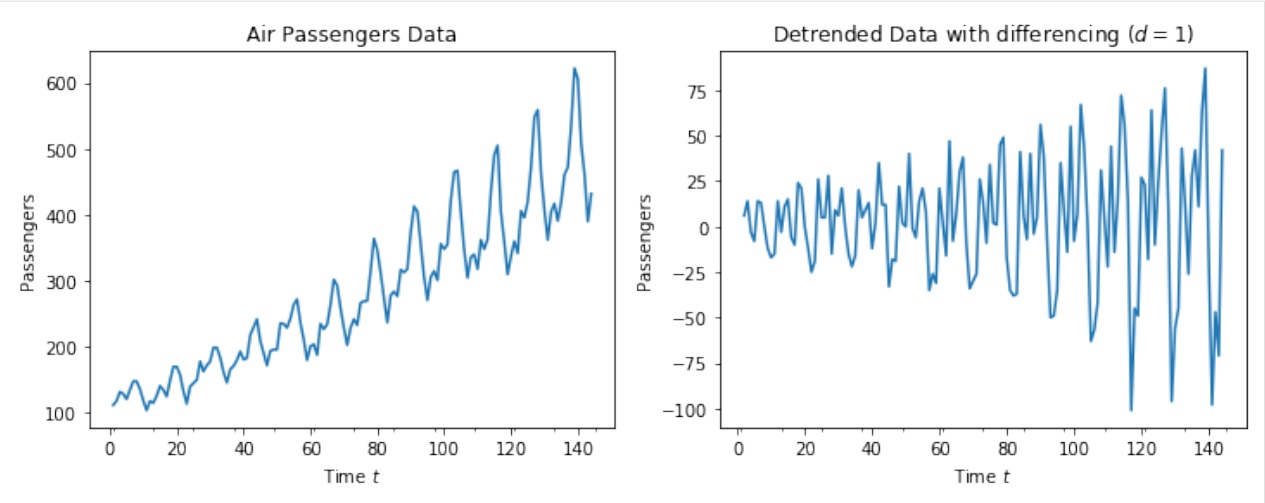
The slope between two points of the time series is the *first-order difference* and we use the notation

$$\Delta^1 x_t = x_t - x_{t-1}.$$

With the first order difference, we detrend time series that have a moving mean value, i.e., $\hat{x}_t = \Delta^1 x_t$. If both the mean value as well as the change of the mean are moving, we can use *second-order differencing*, which is defined as

$$\Delta^2 x_t = \Delta^1 x_t - \Delta^1 x_{t-1} = x_t - 2 \cdot x_{t-1} + x_{t-2}.$$

When we apply first-order differencing to our example, we get the following.

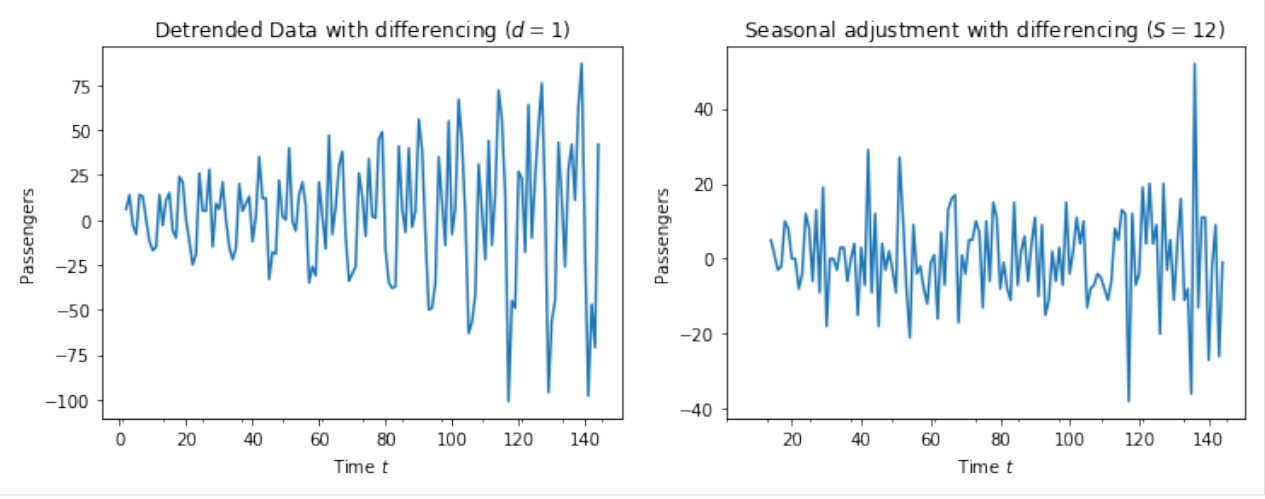


The resulting plot looks centered around 0, but there is still a seasonal fluctuation. We observe that the range of the y-axis is only $75 - (-100) = 175$, i.e., about two thirds of the range of the time series values is explained by the trend.

Differencing can also be used for seasonal adjustment. In this case, we do not use the difference between two points next to each other, but rather between two consecutive points in time during the season. Thus, if we have a length of a season of s, we can calculate our seasonal adjustment as $\hat{\hat{x}} = \Delta_s \hat{x}_t$ with

$$\Delta_s \hat{x}_t = \hat{x}_t - \hat{x}_{t-s}.$$

When we apply seasonal differencing with a length of the season of $s = 12$, we get the following.



There is no pattern left in the plot on the right. The fluctuations around zero look completely random, which is a strong indication that the time series is now stationary. Moreover, the range of the $y - axis$ is further reduced to only $40 - (-40) = 80$, i.e., about half magnitude of the changes in the time series after detrended are explained through the seasonality. The remainder is the autocorrelation of the time series, that must still be modeled.

9.3.3 Comparison of the Approaches

The two methods for modeling the trend and seasonality both have advantages and weaknesses. The advantage of regression and seasonal means is that we get interpretable values through the slope of the regression and the monthly mean values. However, this method is limited, because it only works for linear trends and with seasonal effects that have no trend. Thus, the data may not be fully stationary with this approach.

This is the strength of differencing: it can deal with both changes in the mean, as well as changes in the movement of the mean. This also holds true for the seasonal adjustment, which could also have higher orders, even though we only showed first-order seasonal adjustment. Thus, the likelihood of getting stationary data is higher with this approach. However, we get no interpretable values for the trend or seasonality that quantify the effects.

9.4 Autocorrelation with ARMA

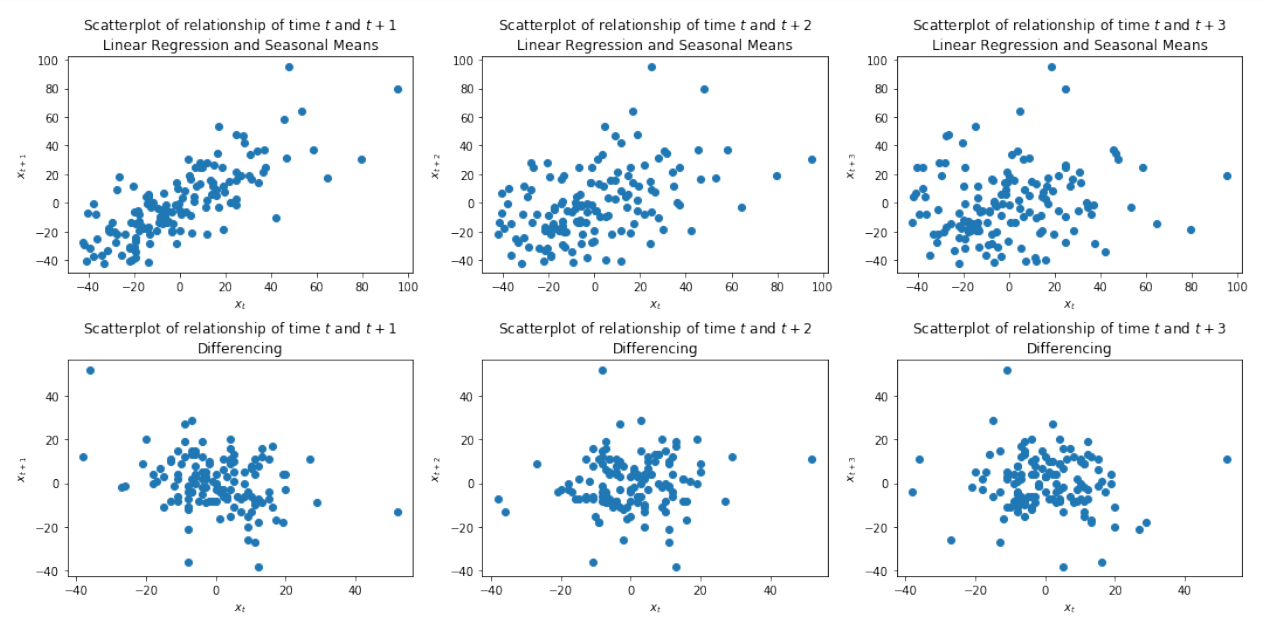
The last aspect of a time series that we must model is the autocorrelation.

9.4.1 Autocorrelation and Partial Autocorrelation

The autocorrelation is the direct relationship of the values of the time series at different points in time. Thus, we have the correlation between the time series with itself.

> Note:
>
> *Auto* means *self* in Latin, i.e., we have the Latin meaning and not the short form of automatic. If these are mixed up in this case, this does not make sense and can be confusing.

What the autocorrelation is can be visualized with scatterplots. Below, we visualize the relationship between the time steps of the time series for our running example with both variants of detrending and seasonal adjustment.



The plots in the first column show the relationship with the next time step, the second column two time steps into the future, and the third column three time steps into the future. With the Linear Regression and the seasonal means, we see that there is a linear relationship between x_t and $x_t + 1$, which gets weaker with x_{t+2} and vanishes with x_{t+3}. This relationship is the autocorrelation of the time series. In the second row, we see that there does not seem to be any

autocorrelation within the time series where we used differencing. The relationship between the different time steps looks completely random.

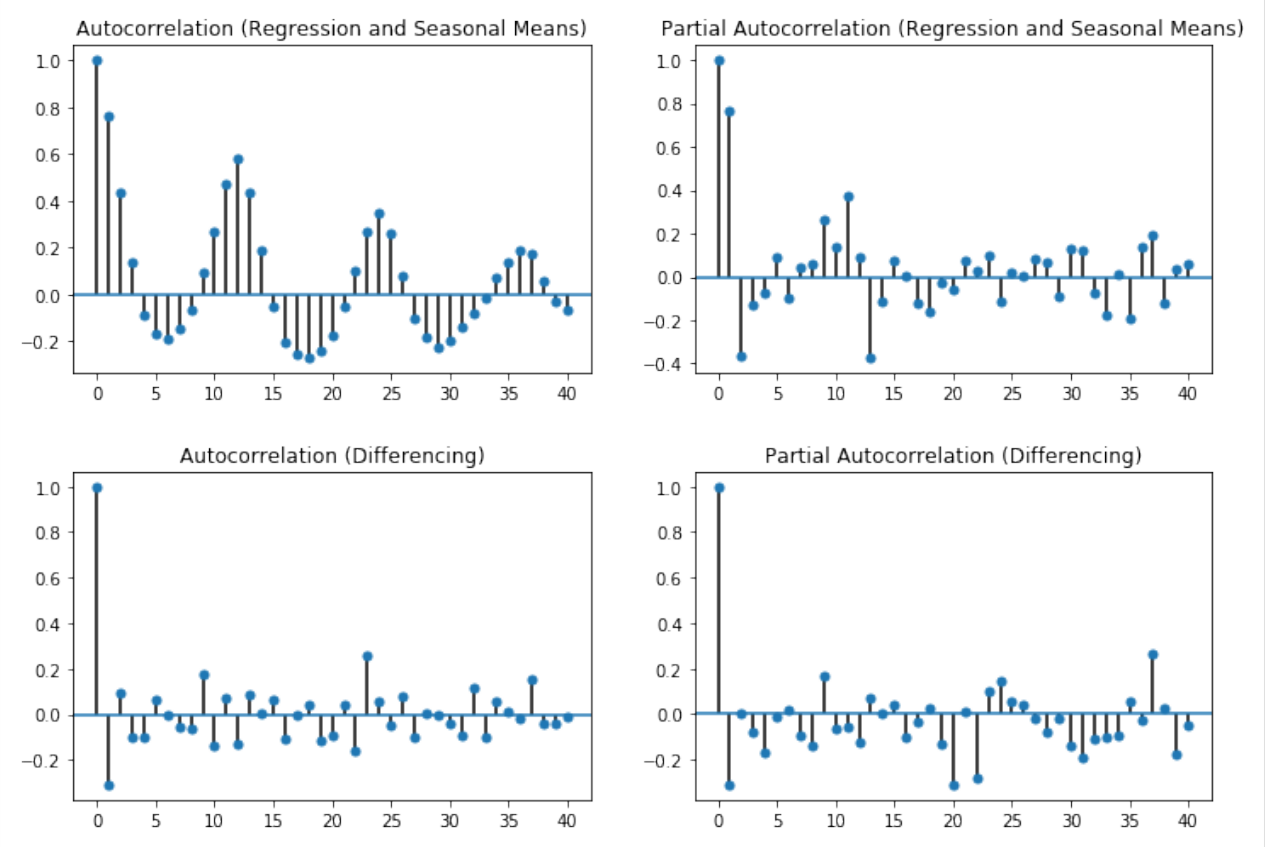
The plots of the autocorrelation above ignore one important aspect:

- x_{t+2} is correlated with x_{t+1}
- x_{t+1} is correlated with x_t
- How much of the correlation between x_{t+2} and x_t is already explained by the indirect correlation between x_{t+2} and x_t though x_{t+1}?

The issue is that there is *carry over* between the different points in the time series. While this is hard to parse with the mathematical notation, the meaning of this gets clearer with an example. If the number of passengers in March are correlated with the value in February, and the value in February is correlated with the value in January, then the value of January has an indirect impact on March, because it has an impact on February. Thus, some of the correlation between January and February carries over to March.

However, there may also be *direct* correlation between January and March. This direct correlation is called *partial autocorrelation*. The partial autocorrelation is the autocorrelation without the carryover, i.e., only the direct correlation.

The plots below show the autocorrelation and the partial autocorrelation of our example.



The bars in the plots indicate the strength of the correlation. While there is no fixed guidelines with respect to what is a strong or a weak correlation, commonly used convention is that absolute values below 0.3 indicate that there is no correlation, between 0.3 and 0.5 that there is a weak correlation, between 0.5 and 0.7 that there is a moderate correlation, and above 0.7 that there is a strong correlation. In the following, we are only interested in at least moderate correlations.

The first row of the plot shows the autocorrelation and partial autocorrelation for the time series that we detrended with the regression and the seasonal means. The autocorrelation shows a clear pattern of alternating positive and negative

correlations over time with a periodicity of 12 time steps. This is the residual seasonal effect, that we can observe within the autocorrelations. The partial autocorrelations show that there is a strong direct influence of x_{t+1} and also that the time steps between $t + 10$ and $t + 13$ have a weak direct influence on x_t that is not explained through the carry over. This is, again, the residual seasonal effect, that we have not corrected.

The second row of the plot shows the autocorrelation and partial autocorrelation for the time series where we used differencing for the adjustments. As we already saw above, there is no correlation between the time steps, both in the autocorrelation and the partial autocorrelation. This observation is the same as we had with the scatterplots.

9.4.2 AR, MA and ARMA

What we are still missing is a way to model the autocorrelation. An common approach is to use *AutoRegressive* (AR) or *Moving Average* (MA) models. If both are used together, this is an ARMA model.

The AR models model a linear relationship between the timesteps. They model the *direct* influence of the past points p in time on the current point in time as

$$c + \epsilon_t + \sum_{i=1}^{p} a_i x_{t-i}$$

where $c \in \mathbb{R}$ is a constant change over time, ϵ_t is *white noise*, and $a_i \in \mathbb{R}, i = 1, ..., p$ are coefficients that determine the influence past values of the time series. Thus, the AR model combines a fixed change c, with a random change ϵ_t and the last p values of the time series.

> Note:
>
> White noise are random variables with an expected value of 0. This means that in the expectation, the white noise does not influence the mean value of time series. Usually, the white noise is modeled by normal distributions with a mean value of 0 and a small standard deviation.

The MA models are *random*, i.e., they are used to model how random effects influence the values of the time series and are defined as

$$c + \epsilon_t + \sum_{j=1}^{q} b_j \epsilon_{t-j}$$

c and ϵ_t are the same as for the AR model. The second part of the MA model ($\sum_{j=1}^{q} b_j \epsilon_{t-j}$ is the influence of the past noise on the next value. For example, a random increase in air passengers in January may mean that there is a decrease of passengers in February. This would be modeled with a negative coefficient for b_1 in the MA model.

When we combine the AR with the MA, we get the ARMA model for the autocorrelation, which is defined as

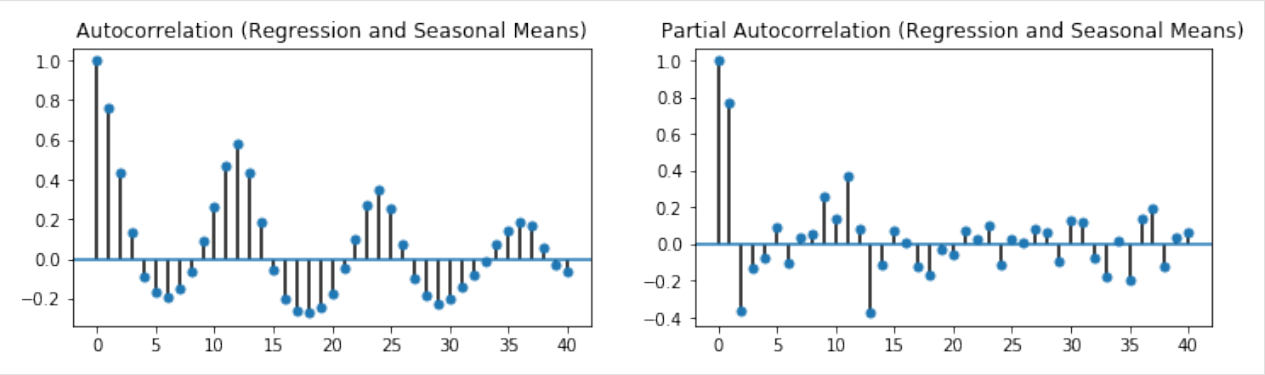
$$x_t = c + \epsilon_t + \sum_{i=1}^{p} a_i x_{t-i} + \sum_{j=1}^{q} b_j \epsilon_{t-j}.$$

In the ARMA model we combine the influence of past values through the coefficients of the AR part of the model, with the influence of random fluctuations.

9.4.3 Picking p and q

What is missing is how the values p and q of an ARMA model are selected, i.e., how many past time steps influence the values of the time series. This is what the plots of the autocorrelation and partial autocorrelation are used for.

For the time series that we detrended with linear regression and seasonal means, we have the following (partial) autocorrelations.



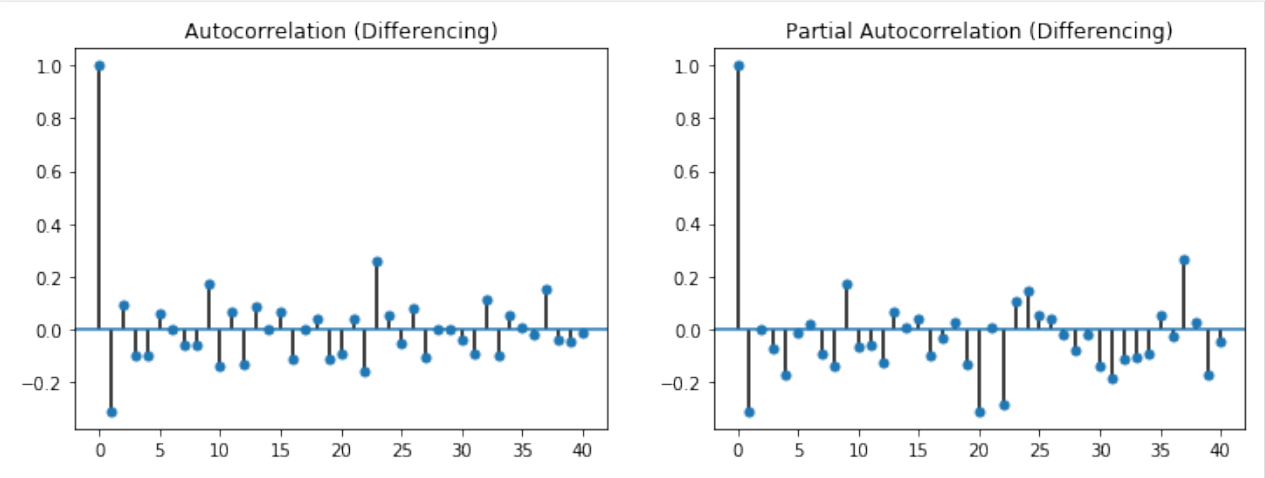
Since the AR part of the model is the direct influence of past values of the time series, we can use the partial autocorrelation to estimate suitable values. In this case, $p = 1$ or $p = 13$ would be suitable, as this would capture all values that seem to have a direct impact on the current estimate. With $p = 1$ we have a small model, whereas we would get a relatively large model with $p = 13$. The large model should be more accurate, because residual seasonal effect is captured, but the model would not be easy to interpret. Moreover, there would be a higher risk of overfitting.

For the MA part, we are interested in how long we can see the effect of random fluctuations in the model. For time series that can be fully described with an MA term, the value of q can be determined with the autocorrelation, because this tends towards zero after q steps. Thus, we could just look at when the autocorrelation goes towards zero and use this for q. This seems to be the case after three full seasons, i.e., at $q = 36$. However, this ignores the direct impact of the prior values of the time series of the AR part. For example, the autocorrelation between at $t + 1$ is also covered by the partial autocorrelation, most of the autocorrelation at time $t + 12$ as well. If we account for this, the impact of the random fluctuations is relatively small. Thus, a suitable q could also be $q = 1$ or $q = 12$, depending on the choice of p.

Together, this means that there are three good candidates for p and q:

- $p = 1, q = 1$ in case a small model is desired and it is acceptable that the residual seasonal effect is not fully accounted for.
- $p = 1, q = 12$ to use the MA part of the model to account for the residual seasonal effect.
- $p = 12, q = 1$ to use the AR part of the model to account for the residual seasonal effect.

For the time series after detrending and seasonal adjustment with differencing, this is simpler, because we have the following (partial) autocorrelations.



In this case, we observe no strong partial autocorrelations, i.e., $p = 0$ because there is no direct impact of the past values on the current values. In the autocorrelations, we see that the first value is with about -0.3 notable, afterwards this tends to zero. As we explained above, this means that $q = 1$ is a suitable choice.

9.4.4 ARIMA

Often, you will also find ARIMA models for time series analysis. ARIMA models have the parameter p, d, q. ARIMA is the same as ARMA, where d-th order differencing is applied to the data. Thus, ARIMA is the same as detrending with differencing and then creating an ARMA model.

CHAPTER

TEN

TEXT MINING

10.1 Overview

Text mining is the application of the techniques we discussed so far to textual data with the goal to infer information from the data. Examples for text mining applications are the analysis of customer reviews to infer their sentiment or the automated grouping of related documents. The problem with analyzing natural language text is that sentences or longer texts are neither numeric nor categorical data. Moreover, there is often some inherent structure in texts, e.g., headlines, introductions, references to other related content, or summaries. When we read text, we automatically identify these internal structures that textual data has. This is one of the biggest challenges of text mining: finding a good representation of the text such that it can be used for machine learning.

For this, the text has to be somehow *encoded* into numeric or categorical data with as little loss of information as possible. The ideal encoding captures not only the words, but also the meaning of the words in their *context*, the grammatical structure, as well as the broader context of the text, e.g., of sentences within a document. To achieve this is still a subject of ongoing research. However, there were many advancements in recent years that made text mining into a powerful, versatile, and often sufficiently reliable tool. Since text mining itself is a huge field, we can only scratch the surface of the topic. The goal is that upon finishing this chapter, you have a good idea the challenges of text mining, know basic text processing techniques, and also have a general idea of how more advanced text mining works.

We will use the following eight tweets from Donald Trump as an example for textual data to demonstrate how text mining works in general. All data processing steps are done with the goal to prepare the text such that it is possible to analyze the topic of the tweets.

```
Oct 4, 2018 08:03:25 PM Beautiful evening in Rochester,
Minnesota. VOTE, VOTE, VOTE! https://t.co/SyxrxvTpZE [Twitter for
iPhone]

Oct 4, 2018 07:52:20 PM Thank you Minnesota - I love you!
https://t.co/eQC2NqdIil [Twitter for iPhone]

Oct 4, 2018 05:58:21 PM Just made my second stop in Minnesota for
a MAKE AMERICA GREAT AGAIN rally. We need to elect @KarinHousley
to the U.S. Senate, and we need the strong leadership of
@TomEmmer, @Jason2CD, @JimHagedornMN and @PeteStauber in the U.S.
House! [Twitter for iPhone]

Oct 4, 2018 05:17:48 PM Congressman Bishop is doing a GREAT job!
He helped pass tax reform which lowered taxes for EVERYONE! Nancy
Pelosi is spending hundreds of thousands of dollars on his
opponent because they both support a liberal agenda of higher
taxes and wasteful spending! [Twitter for iPhone]
```

(continues on next page)

(continued from previous page)

```
Oct 4, 2018 02:29:27 PM "U.S. Stocks Widen Global Lead"
https://t.co/Snhv08ulcO [Twitter for iPhone]

Oct 4, 2018 02:17:28 PM Statement on National Strategy for
Counterterrorism: https://t.co/ajFBg9Elsj https://t.co/Qr56ycjMAV
[Twitter for iPhone]

Oct 4, 2018 12:38:08 PM Working hard, thank you!
https://t.co/6HQVaEXH0I [Twitter for iPhone]

Oct 4, 2018 09:17:01 AM This is now the 7th. time the FBI has
investigated Judge Kavanaugh. If we made it 100, it would still
not be good enough for the Obstructionist Democrats. [Twitter for
iPhone]
```

10.2 Preprocessing

Through preprocessing, text is transformed into a representation that we can use for machine learning algorithms, e.g., for the classification or for the grouping with clustering.

10.2.1 Creation of a Corpus

The first preprocessing step is to create a *corpus* of *documents*. In the sense of the terminology we have used so far, the documents are the objects that we want to reason about, the corpus is a collection of objects. In our Twitter example, the corpus is a collection of tweets, and each tweet is a document. In our case, we already have a list of tweets, which is the same as a corpus of documents. In other use cases, this can be more difficult. For example, if you crawl the internet to collect reviews for a product, it is likely that you find multiple reviews on the same Web site. In this case, you must extract the reviews into separate documents, which can be challenging.

10.2.2 Relevant Content

Textual data, especially text that was automatically collected from the Internet, often contains irrelevant content for a given use case. For example, if we only want to analyze the topic of tweets, the timestamps are irrelevant. It does also not matter if a tweet was sent with an iPhone or a different application. Links are a tricky case, as they may contain relevant information, but are also often irrelevant. For example, the URL of this page contains relevant information, e.g., the author, the general topic, and the name of the current chapter. Other parts, like the http are irrelevant. Other links are completely irrelevant, e.g., in case link shorteners are used. In this case a link is just a random string.

When we strip the irrelevant content from the tweets, we get the following.

```
Beautiful evening in Rochester, Minnesota. VOTE, VOTE, VOTE!

Thank you Minnesota - I love you!

Just made my second stop in Minnesota for a MAKE AMERICA GREAT
AGAIN rally. We need to elect @KarinHousley to the U.S. Senate,
and we need the strong leadership of @TomEmmer, @Jason2CD,
```

(continues on next page)

(continued from previous page)

```
@JimHagedornMN and @PeteStauber in the U.S. House!

Congressman Bishop is doing a GREAT job! He helped pass tax
reform which lowered taxes for EVERYONE! Nancy Pelosi is spending
hundreds of thousands of dollars on his opponent because they
both support a liberal agenda of higher taxes and wasteful
spending!

"U.S. Stocks Widen Global Lead"

Statement on National Strategy for Counterterrorism:

Working hard, thank you!

This is now the 7th. time the FBI has investigated Judge
Kavanaugh. If we made it 100, it would still not be good enough
for the Obstructionist Democrats.
```

What is relevant and irrelevant can also depend on the context. For example, a different use case for Twitter data would be to analyze if there are differences between tweets from different sources. In this case, the source cannot be dropped, but would be needed to divide the tweets by their source. Another analysis of Twitter data may want to consider how the content of tweets evolves over time. In this case, the timestamps cannot just be dropped. Therefore, every text mining application should carefully consider what is relevant and tailor the contents of the text to the specific needs.

10.2.3 Punctuation and Cases

When we are only interested in the topic of documents, the punctuation, as well as the cases of the letters are often not useful and introduce unwanted differences between the same words. A relevant corner case of dropping punctuation and cases are acronyms. The acronym `U.S.` from the tweets is a perfect example for this, because this becomes `us`, which has a completely different meaning. If you are aware that the may be such problems within your data, you can manually address them, e.g., by mapping and `US` to `usa` after dropping the punctuation, but before lower casing the string.

When we apply this processing to our tweets, we get the following.

```
beautiful evening in rochester minnesota vote vote vote

thank you minnesota  i love you

just made my second stop in minnesota for a make america great
again rally we need to elect karinhousley to the usa senate and
we need the strong leadership of tomemmer jason2cd jimhagedornmn
and petestauber in the usa house

congressman bishop is doing a great job he helped pass tax reform
which lowered taxes for everyone nancy pelosi is spending
hundreds of thousands of dollars on his opponent because they
both support a liberal agenda of higher taxes and wasteful
spending

usa stocks widen global lead
```

(continues on next page)

(continued from previous page)

```
statement on national strategy for counterterrorism

working hard thank you

this is now the 7th time the fbi has investigated judge kavanaugh
if we made it 100 it would still not be good enough for the
obstructionist democrats
```

10.2.4 Stop Words

Another aspect of text is that not every word carries relevant meaning. Many words are required for correct grammar, but do not modify the meaning. Examples for such words are `the`, `a`, and `to`. Moreover, such words occur in almost any sentence. Other examples for frequently occuring words are `I`, `we`, `is` and `too`. For many text mining approaches, such words are not well suited, because they cannot be used to differentiate between documents. For example, these words are usually not important to specify the topic or sentiment of a document. Thus, a common preprocessing step is to remove such words. While it is possible to remove stop words with a manually defined list, there are also lists of such words that can be used.

When we apply a list of English stop words to the tweets, we get the following.

```
beautiful evening rochester minnesota vote vote vote

thank minnesota love

made second stop minnesota make america great rally need elect
karinhousley usa senate need strong leadership tomemmer jason2cd
jimhagedornmn petestauber usa house

congressman bishop great job helped pass tax reform lowered taxes
everyone nancy pelosi spending hundreds thousands dollars
opponent support liberal agenda higher taxes wasteful spending

usa stocks widen global lead

statement national strategy counterterrorism

working hard thank

7th time fbi investigated judge kavanaugh made 100 would still
good enough obstructionist democrats
```

10.2.5 Stemming and Lemmatization

Words have different spellings, depending on their grammatical context, e.g., whether something is used in the singular or the plural. Moreover, sometimes there are related verbs, adjectives, and nouns. There are also synonyms, i.e., multiple words with the same meaning. For our text mining, this means that we would observe all these different natural language terms for the same context as different words and could not easily identify that they actually mean the same. Stemming and lemmatization are a potential solution for this.

With stemming, words are reduced to their stem. For example, the terms `spending` and `spends` are reduced to their *stem*, which is `spend`. Stemming usually works with an algorithmic approach, e.g., using Porter's stemming algorithm and is limited to shorting words to their stem. Other aspects, e.g., the harmonization of `good` and `well` cannot be done with stemming.

Lemmatization is an approach for the harmonization of words based on word lists. The wordlists contain definitions for which terms can be used synonymously. Through lemmatization, one representative of this wordlist is chosen that replaces all other words with the same meaning. This way, `good` can be used to replace all usages of `well` to harmonize the language.

When we want to apply these techniques, we should first apply the lemmatization, then the stemming. The reason for this is that the word stems created by stemming may not be part of the lemmatization dictionary. Therefore, we would reduce the power of the lemmatization, if we first stem the words.

When we apply these techniques to our tweets, we get the following. First, we lemmatize the tweets.

```
beautiful evening rochester minnesota vote vote vote

thank minnesota love

made second stop minnesota make america great rally need elect
karinhousley usa senate need strong leadership tomemmer jason2cd
jimhagedornmn petestauber usa house

congressman bishop great job helped pas tax reform lowered tax
everyone nancy pelosi spending hundred thousand dollar opponent
support liberal agenda higher tax wasteful spending

usa stock widen global lead

statement national strategy counterterrorism

working hard thank

7th time fbi investigated judge kavanaugh made 100 would still
good enough obstructionist democrat
```

We can see that some words are shortened, e.g., `stocks` became `stock`. When we apply stemming, we get the following.

```
beauti even rochest minnesota vote vote vote

thank minnesota love

made second stop minnesota make america great ralli need elect
karinhousley usa senat need strong leadership tomemm jason2cd
```

(continues on next page)

(continued from previous page)

```
jimhagedornmn petestaub usa hous

congressman bishop great job help pa tax reform lower tax everyon
nanci pelosi spend hundr thousand dollar oppon support liber
agenda higher tax wast spend

usa stock widen global lead

statement nation strategi counterterror

work hard thank

7th time fbi investig judg kavanaugh made 100 would still good
enough obstructionist democrat
```

We observe that most words are now reduced to their stem and that these stems are often no real words anymore. This is a major difference between stemming and lemmatization. The results of lemmatization will always be words that can still be found in a dictionary, while stemming creates sometimes hard-to-read representations of words.

10.2.6 Visualization of the Preprocessing

We can use wordclouds to demonstrate the impact of each preprocessing step. Wordclouds visualize textual data and determine the size of the displayed words through their count in the data.

10.2.7 Bag-of-Words

Once we have preprocessed the text, we can create a numeric representation in form of a *bag-of-words*. A bag of words is similar to a one-hot-encoding of the text: each unique word is a separate feature. The value of the features for a document is the count of that word within the documented. This is also called the *term frequency* (TF). This is where the harmonization of the words pays off: now similar words increase the count for the same feature and not two different features. For example, the bag of words for our tweets looks like this.

```
   100  7th  agenda  america  beautiful  bishop  congressman  \
0    0    0       0        0          1       0            0
1    0    0       0        0          0       0            0
2    0    0       0        1          0       0            0
3    0    0       1        0          0       1            1
4    0    0       0        0          0       0            0
5    0    0       0        0          0       0            0
```

(continues on next page)

(continued from previous page)

```
6    0    0        0         0           0       0             0
7    1    1        0         0           0       0             0

   counterterrorism  democrat  dollar  ...  thank  thousand  time  tomemmer  \
0                 0         0       0  ...      0         0     0         0
1                 0         0       0  ...      1         0     0         0
2                 0         0       0  ...      0         0     0         1
3                 0         0       1  ...      0         1     0         0
4                 0         0       0  ...      0         0     0         0
5                 1         0       0  ...      0         0     0         0
6                 0         0       0  ...      1         0     0         0
7                 0         1       0  ...      0         0     1         0

   usa  vote  wasteful  widen  working  would
0    0     3         0      0        0      0
1    0     0         0      0        0      0
2    2     0         0      0        0      0
3    0     0         1      0        0      0
4    1     0         0      1        0      0
5    0     0         0      0        0      0
6    0     0         0      0        1      0
7    0     0         0      0        0      1

[8 rows x 70 columns]
```

We can use the word counts in the bag of words as input for learning algorithms. For clustering, this would mean that we group based on documents that use the same words, for classification this would mean that we compute scores and classes based on how often words occur.

10.2.8 Inverse Document Frequency

One popular addition to the bag-of-words is to use the *inverse document frequency* (IDF). The idea behind this approach is to weight words by their *uniqueness* within the corpus. If a word occurs only in few documents, it is very specific and should have a stronger influence than words that occur in many documents. The inverse document frequency is related to stop word removal. However, instead of removing words that occur in many documents, they just get a lower weight in comparison to words that occur in only few documents through the IDF, which is defined as

$$IDF_t = \log \frac{N}{D_t}$$

where t is a word (term), N is the number of documents in the corpus and D_t is the number of documents in which t appears. The TFIDF combines the term frequency with the weights of the inverse document frequency and can be used instead of the to replace the TF in the bag of words. The definition of TFIDF for a word t is

$$TFIDF_t = TF_t \cdot IDF_t.$$

For our tweets, we get the following when we use the TFIDF.

```
        100       7th    agenda   america  beautiful    bishop  congressman  \
0  0.000000  0.000000  0.000000  0.000000   0.283823  0.000000     0.000000
1  0.000000  0.000000  0.000000  0.000000   0.000000  0.000000     0.000000
2  0.000000  0.000000  0.000000  0.208439   0.000000  0.000000     0.000000
3  0.000000  0.000000  0.175141  0.000000   0.000000  0.175141     0.175141
```

(continues on next page)

(continued from previous page)

```
4  0.000000  0.000000  0.000000  0.000000   0.000000  0.000000     0.000000
5  0.000000  0.000000  0.000000  0.000000   0.000000  0.000000     0.000000
6  0.000000  0.000000  0.000000  0.000000   0.000000  0.000000     0.000000
7  0.271158  0.271158  0.000000  0.000000   0.000000  0.000000     0.000000

   counterterrorism  democrat    dollar  ...     thank  thousand      time  \
0               0.0  0.000000  0.000000  ...  0.000000  0.000000  0.000000
1               0.0  0.000000  0.000000  ...  0.546004  0.000000  0.000000
2               0.0  0.000000  0.000000  ...  0.000000  0.000000  0.000000
3               0.0  0.000000  0.175141  ...  0.000000  0.175141  0.000000
4               0.0  0.000000  0.000000  ...  0.000000  0.000000  0.000000
5               0.5  0.000000  0.000000  ...  0.000000  0.000000  0.000000
6               0.0  0.000000  0.000000  ...  0.480535  0.000000  0.000000
7               0.0  0.271158  0.000000  ...  0.000000  0.000000  0.271158

   tomemmer       usa     vote  wasteful     widen   working     would
0  0.000000  0.000000  0.85147  0.000000  0.000000  0.000000  0.000000
1  0.000000  0.000000  0.00000  0.000000  0.000000  0.000000  0.000000
2  0.208439  0.323043  0.00000  0.000000  0.000000  0.000000  0.000000
3  0.000000  0.000000  0.00000  0.175141  0.000000  0.000000  0.000000
4  0.000000  0.361285  0.00000  0.000000  0.466228  0.000000  0.000000
5  0.000000  0.000000  0.00000  0.000000  0.000000  0.000000  0.000000
6  0.000000  0.000000  0.00000  0.000000  0.000000  0.620116  0.000000
7  0.000000  0.000000  0.00000  0.000000  0.000000  0.000000  0.271158

[8 rows x 70 columns]
```

10.2.9 Beyond the Bag-of-Words

While the bag-of-words is a good technique that scales well, there are also some severe limitations. One such limitations are that the bag-of-words does not account for the structure of documents, i.e., grammar and the context are completely ignored. Another one is that the similarity between words is ignored. For example, it makes sense that `dollar` and `euro` are two separate terms in the bag-of-words. However, the words are similar to each other, because both are currencies, which may be relevant for the text mining.

There are also more complex techniques that can be used. A simple extension of the bag-of-words that accounts at least to some degree for the context and structure is to use n-grams instead of words. A n-gram consists of subsequences of n words. Thus, we do not count how often words occur anymore, but rather how often sequences of words occur. The drawbacks of n-grams is that the chance that it is unlikely that n-grams for larger values of n occur more than once, because this would require the exact same wording. Thus, n-grams may be needlessly specific and, therefore, hinder generalization. Moreover, the number of n-grams growths exponentially with n. Thus, this approach only works to capture the direct context of a word.

A recent approach is to use *word embeddings* instead of the bag-of-words. Word embeddings transform words into a high dimensional space (e.g., the $\mathbb{R}^d$) such that similar words are close to each other in the embedded space. State-of-the-art word embeddings like BERT can also take the context of words into account and, thereby, further improve the similarity. The word embeddings themselves are learned through deep neural networks with a *transformer* architecture using huge amounts of textual data.

10.3 Challenges

The difficulties with text mining do not stop, just because we now have a numeric representation of the text that we can use as input for algorithms. There are many other problems, many of them not yet solved, that make text mining into such a hard problem.

10.3.1 Dimensionality

The first problem is the dimensionality. In our example of just eight tweets, we already have 70 different terms after preprocessing. In longer text, there are several thousand unique terms, even after preprocessing. Thus, there is a huge number of categorical features. However, not only the number of features may be very large, the number of documents can also be very large. For example, there are already over 50,000 tweets by Donald Trump (June 2020), and hundreds of millions of tweets every day.

The combination of many features and many instances means that text mining can consume huge amounts of runtime. Therefore, large text mining applications only work with efficiently implemented algorithms, often on dedicated hardware that allows for massive parallelization. These requirements are also the reason why an algorithm that scales well with large amounts of data, like Multinomial Naive Bayes, is still a popular choice for text mining.

10.3.2 Ambiguities

The second problem is that natural language if often ambiguous. Partially, this cannot be solved, because natural language is, unless used with great care, imprecise and the exact meaning is often only understandable from a very broad context. For example, the following sentence can have a different meaning, that can only be understood from the context.

- I hit a man with a stick. (I used a stick to hit a man.)
- I hit a man with a stick. (I hit the man who was holding a stick.)

Such problems are often impossible to resolve and lead to noise in the analysis.

Other problems with ambiguities are to some degree solvable, but may also lead to noise in the analysis, especially with a bag-of-words. *Homonyms* are words with multiple meanings. For example, *break* can mean *to break something*, but also *take a break*. The bag-of-words does not differentiate between these meanings. Other approaches, such as n-grams and word embeddings can, to some degree, account for this, but not perfectly.

There are also syntactic ambiguities, where the meaning changes depending on the syntactic structure of a sentence. The following nine-word sentence uses the word *only* in every possible position. The meaning changes with every position.

- Only he told his mistress that he loved her. (Nobody else did.)
- He only told his mistress that he loved her. (He didn't show her.)
- He told only his mistress that he loved her. (Kept it a secret from everyone else.)
- He told his only mistress that he loved her. (Stresses that he had only ONE!)
- He told his mistress only that he loved her. (Didn't tell her anything else.)
- He told his mistress that only he loved her. ("I'm all you got, nobody else wants you.")
- He told his mistress that he only loved her. (Not that he wanted to marry her.)
- He told his mistress that he loved only her. (Yeah, don't they all...)
- He told his mistress that he loved her only. (Similar to above one.)

10.3.3 And Many More

Above, we discussed some examples in detail. There are many others, for example the following.

- Bad spelling which leads to unknown words.
- The evolution of language, especially in form of new words or slang.
- Imperfection of preprocessing approaches, e.g., synonyms that are not part of the wordlist for lemmatization.
- The parsing of text, which have different encodings and character sets (Chinese, Japanese, Korean, Arabic, Latin, . . .)

This also means that text mining applications are, at least currently, never finished, because there are always more unsolved problems with textual data. Users of text mining must be aware of this, because otherwise projects may never finish. At some point, a decision must be made that the remaining problems are not within the scope of a project and the performance of a text mining approach is sufficient.

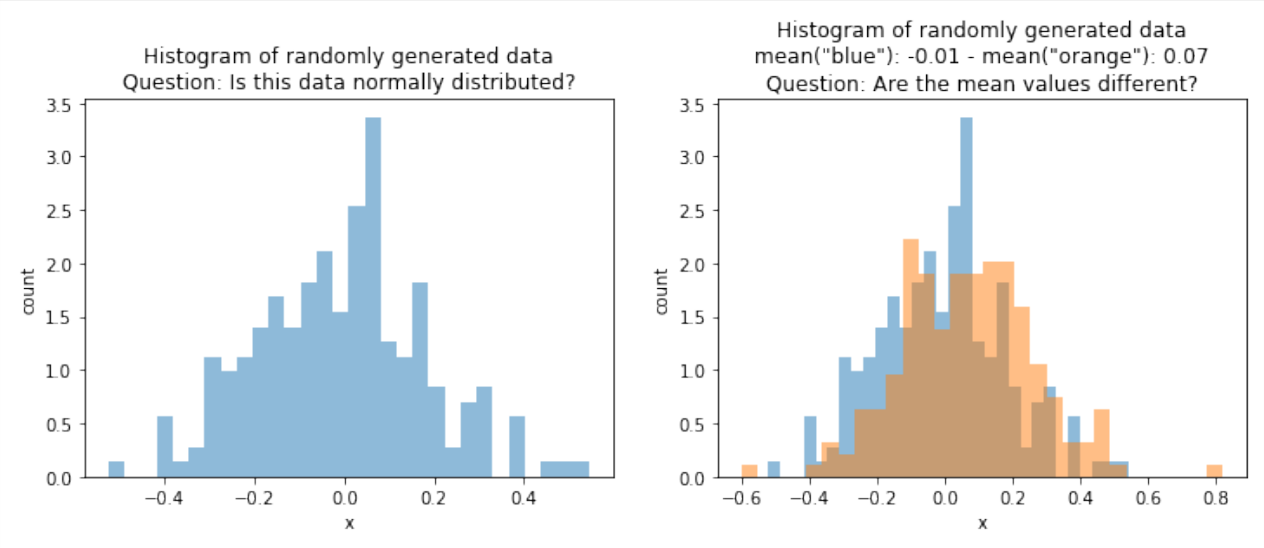
CHAPTER

ELEVEN

STATISTICS

11.1 Motivation

So far, we have often looked at data. For example, in Chapter 3 we visually analyzed the distributions of data. Below are two histograms of data that we could also visually analyze.



The histogram on the left looks like it has a bell shape, which indicates that the data may be normal. However, we cannot be sure, without additional analysis. It would also be possible that the data just randomly looks like that, i.e., is some outlier and that the data is not really normal.

The histogram on the right shows two samples $X_1 = \{x_1^1, ..., x_{n_1}^1\}$ ("blue") and $X_2 = \{x_1^2, ..., x_{n_2}^2\}$ ("orange"). Both look similar, but it looks like the mean value of the blue data may be a bit smaller than that of the orange data. However, this could also just be a random effect, and there is no real difference.

Statistics gives us the methods we need to get (probabilistic) answers to these questions. Within this Chapter, we give a short introduction into the (quite large) world of statistics. Upon completion of this chapter, you should know what hypothesis testing is, how this can be used correctly, and why hypothesis testing is often not useful, if used without additional statistical tools like effect size and confidence intervals.

11.2 Hypothesis Testing

Hypothesis testing is the primary tool of *frequentist statistics*. The idea behind hypothesis testing is the following. If we have certain assumptions on the data, we can formulate these assumptions as an hypothesis about the data. We always have two hypotheses, the *null hypothesis* and the *alternative hypothesis*.

- Null Hypothesis (H_0): Assumption of the test holds and is failed to be rejected at some level of significance.
- Alternative Hypothesis (H_1, H_a): Assumption of the test does not hold and is rejected at some level of significance.

The difficult part of hypothesis testing is understanding what these abstract concepts mean. What exactly is the assumption of a test? What does "(failed to be) rejected at some level of significance" mean? Since these questions are often misunderstood and misused, we explain them using the example of the t-test.

11.2.1 t-Test

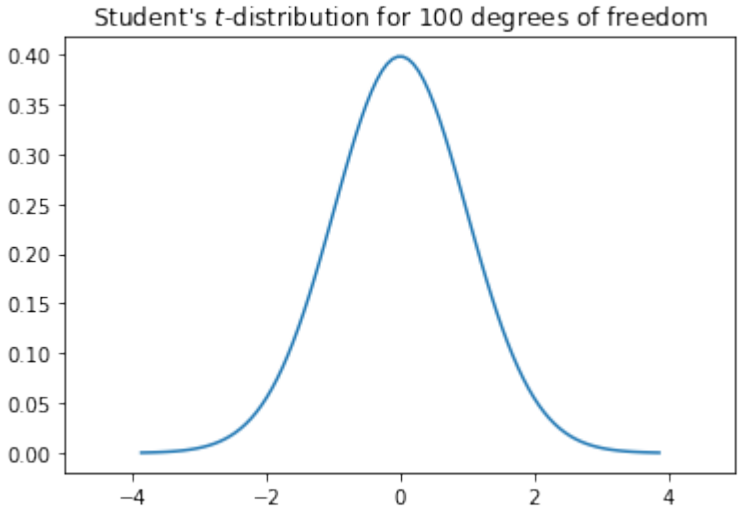
The t-Test can be seen as the father of all statistical tests and was first introduced by William Gosset under the pseudonym "Student". This oldest version of the test and the related probability distribution are called Student's t-Test and Student's distribution. Here, we consider a newer variant of the t-Test, i.e., Welch's t-Test. Welch's t-Test has the following null and alternative hypotheses:

- H_0: The means of two normally distributed populations X_1 and X_2 are equal.
- H_a: X_1 and X_2 are not from two normally distributed populations with equal mean values.

Thus, this test can be used to determine if two normally distributed populations have the same mean value. This is relevant in many use cases, e.g., to answer questions like "does this medication improve the health" or "is my deep neural network better than a random forest". Please note that the alternative hypothesis of the t-Test is not that the mean values are different. Only if both distributions are normal, this would mean that the mean values are different. If one of the populations is not normally distributed, we do not know whether the null hypothesis was rejected because the data was not normal or because the means are different. This misinterpretation of alternative hypothesis is one of the common mistakes when hypothesis testing is used.

In order to evaluate whether the null hypothesis holds, hypothesis testing evaluates *how likely the data is, given the hypothesis*. For the t-Test this means that we want to calculate how likely it is that we observe the populations X_1 and X_2, in case both are normally distributed with the same mean value. This probability is the infamous *p-value*.

To determine the p-value, we need the probability density function of a test statistic. In case of the t-Test, this is Student's t-Distribution.

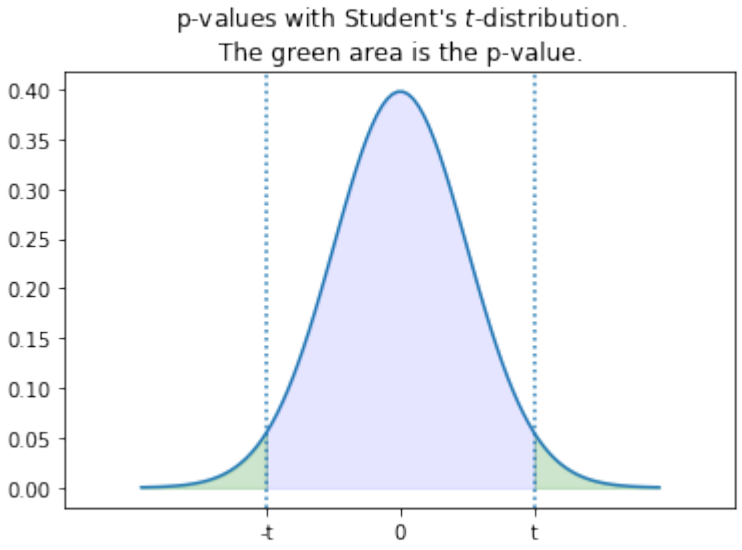


This distribution looks similar to the normal distribution, the difference is that the tails are a bit longer than those of the normal distribution. Student's t-distribution is defined as the distribution of the deviation of the arithmetic mean of a sample drawn from a standard normal distribution with a mean value of 0 and a standard deviation of 1. Thus, the t-Distribution tells us how much deviation of the mean values between two samples drawn from a standard normal distribution we can expect. This is directly related to the likelihood of the data, given the hypothesis that data is normal.

The difference we still have is that the t distribution is for a specific mean value and standard deviation, while our hypothesis is for normal distributions with any mean value and standard distribution. This is why we need a *test statistic*, in case of the t-Test the t value, which is defined as

$$t = \frac{mean(X_1) - mean(X_2)}{\sqrt{\frac{sd(X_1)^2}{n_1} + \frac{sd(X_2)}{n_2}}}$$

A bit simplified, the test statistics t is the rescaled difference between the two populations X_1 and X_2 such that t estimates the difference of the means as if they were two standard normal distributions. Thus, we can use the t value and Student's t-distribution to calculate how probable it is, that these populations are observed, given that the means are not different. For this, we calculate the area under the t-distribution outside of $\pm t$.



The green area in the plot above is the probability, that the mean of a sample drawn from a standard normal distribution deviates by at least t from 0. Since we calculated t such that this is the same as the probability that the means of X_1 and X_2 are different, this is the p-value of the test, hence the probability that the data is observed given the null hypothesis is true.

11.2.2 Significance Level

Since the p-value indicates how likely the data is, given that the null hypothesis is true, it is reasonable to argue that the null hypothesis may not be true, if the p-value is very small, i.e., the data is very unlikely if the null hypothesis were true. This concept is formalized with the *significance level* α that is used to define the *statistical significance* of results.

The significance level is defined as the (acceptable) probability of rejecting the null hypothesis, given that it is true. While the significance level depends on the domains, commonly used values are 0.05, or more recently also 0.005. With the significance level, we can decide if we reject the null hypothesis or not:

- If $p\text{-}value > \alpha$ we fail to reject the null hypothesis. This means that the data can be reasonably well explained, if the null hypothesis were true.
- If $p\text{-}value \leq \alpha$ we reject the null hypothesis and have a *statistically significant result*, e.g., a statistically significant difference of the mean values in case of the t-Test if both populations are normal.

Please note that neither case means with absolute certainty that the null hypothesis or the alternative hypothesis is true. Hypothesis testing is probabilistic. When we fail to reject the null hypothesis, the null hypothesis could still be false, but this is not supported by the data. When we reject the null hypothesis, the null hypothesis could still be true, it is just unlikely - but not impossible! - that such extreme data would be generated. This is also why we use the somewhat complicated wording of "fail to reject" or "reject" at some significance level. Absolute statements of the form "there is a difference", are not supported by hypothesis testing, only probabilistic statements of the form "there is a statistically significant difference with a significance level of α.

11.2.3 Overview of Statistical Tests

The explanations above are similar for all statistical tests: we have a null hypothesis that is based on our assumptions and we calculate a p-value that we can use to either fail to reject or reject the null hypothesis given some level of significance.

Here are the null hypothesis of some important statistical tests.

- Welch's t-Test: The means of two normally distributed populations are equal
- Mann-Whitney-U Test / Wilcoxon-Ranksum-Test: The values from one population dominate the values of another population (more or less difference of means/medians)
- Shapiro-Wilk Test: A population of 3 to 5000 independent samples is normally distributed.
- Levene's Test: The variances of a group of populations are equal.
- ANOVA (F-Test): The mean values of a group normally distributed populations with equal variances are equal.
- Kolmogorov-Smirnoff Test: Two populations have the same probability distribution.

We already know that we can use Welch's t-Test to determine if the means of two normal distributions are significantly different. In case the data is not normal, we can use the Mann-Whitney-U Test instead. We can use the Shapiro-Wilk Test to determine if our samples are normal, e.g., to decide which test to use for the difference in the central tendency. When we have more than one population, we can use ANOVA to test for differences in the central tendency, given the data is normal and all variances are the same. Levene's tests can be used to determine if is this is the case.

In general, we say that a test is *parametric* if the test has assumptions about the distribution of the data (e.g., that the data must be normal), and *non-parametric* if the test does not make such assumption. Welch's t-Test is an example of parametric test, the Mann-Whitney-U Test is the non-parametric counterpart.

The Kolmogorov-Smirnoff Test is one of the most generic statistical tests, because it can test for arbitrary populations of continuous data if they have the same distribution. Thus, this test can be used under nearly any circumstances. Regardless, using this test is some sort of "last resort", because in case of differences, we have no idea what the difference may be. In comparison, if we first conduct Shapiro-Wilk tests to determine that data is normal and then use Welch's t-Test and find that there is a significant difference, we not only know that there likely is a difference between the populations, but also that this difference is due to a difference in mean values.

11.2.4 Using the t-Test

Now that we have discussed the statistical tests in detail, we come back to our motivation and try to answer the question from the beginning of this chapter: are the mean values of the blue and orange samples different? We already learned, that we cannot answer this question absolutely. But we can use statistical tests to answer the question if the mean values of the blue and orange samples are significantly different at a significance level of α. To determine this, we first apply the Shapiro-Wilk test to both samples, to check if they are normal. If both are normal, we use Welch's t-Test, otherwise we use the Mann-Whitney-U test.

```
p-value of Shapiro-Wilk test for "blue" data:   0.8290
The test found that the data sample was normal, failing to reject
the null hypothesis at significance level alpha=0.050.

p-value of Shapiro-Wilk test for "orange" data: 0.2498
The test found that the data sample was normal, failing to reject
the null hypothesis at significance level alpha=0.050.

Both populations normal. Using Welch's t-test.

p-value of Welch's t-tests: 0.000102
The test found that the population means are probably not equal,
rejecting the null hypothesis at significance level alpha=0.050.
```

Thus, the tests indicate that the data is likely normal and that the mean values are likely not exactly the same, meaning that we found a statistically significant difference between the two mean values at a significance level of $\alpha = 0.05$.

11.2.5 Common Misuses of Hypothesis Testing

With hypothesis testing the devil is in the details, i.e., there are many things that can be forgotten or used in an imprecise way, which often leads to overstating the results. There are some big issues that are frequently misinterpreted.

First, people often use hypothesis testing to make binary statements, e.g., "the data is normal" instead of "the statistical test indicates that the data is likely normal". This is a big difference: one leaves room for the expected errors with hypothesis testing, the other is an absolute statement indicating that this is true. That these mistakes happen (a lot!), is understandable. Writing about the results of statistical tests in a correct way is hard, because you have to be careful to avoid binary statements, while still drawing relevant conclusions. Regardless, good scientific practice mandates that this should be avoided.

Second, p-values are misused because many people do not understand what the p-values are. This is why there are many researchers advocating that hypothesis testing and p-values should be abolished, in favor of other methods. There are also many researchers who say that abolishing p-values would not solve the problem that people do not understand statistics, which is the reason for the misuse. The two most important misuses are *scoring* and *p-hacking*.

Scoring means that the p-values are used as scores and the differences between populations are *ranked* using the p-values. This does not make any sense, because the p-values are not the probability that the hypothesis is true, but rather that the data was generated given that the hypothesis is true. Thus, ranking of p-values would mean that we somehow score the data, which is not the intend.

The other problem, p-hacking, often happens inadvertently. As discussed above, the p-value is the probability that the data was generated, given the hypothesis. Thus, if the p-value is 0.05, there would be a 5% probability that this data was generated, given the hypothesis. In other words, there already is the expectation that there are *false positive test results*, i.e., that we sometimes reject the null hypothesis, even if the null hypothesis is true. In general, this is not a problem, if we keep this probability low. However, if you conduct 20 tests and each test has a 5% probability that the test is a false positive, it is almost certain that there are false positive results. This effect can be countered by *adjusting* the p-value, e.g., with Bonferroni correction.

Such p-hacking also often happens inadvertently through *subgroup analysis*. A famous example of this is this article, where the authors were forced to conduct subgroup analysis that were not planned. To demonstrate that this likely leads to false positive results, they included an analysis of the impact of the zodiac signs on the treatment.

11.3 Effect Sizes

An often-forgotten aspect of statistical tests is that significant is not the same as meaningful. In our example, we have two normal distributions with the following arithmetic mean and standard deviation.

```
mean ("blue")   = -0.01
sd   ("blue")   = 0.19
mean ("orange") = 0.07
sd   ("orange") = 0.20
```

This means that we have a difference of means that is relatively small in comparison to the standard deviation. The *effect size* extends the context of significance with a notion of how large the difference actually is. For normally distributed data, the effect size for the difference between the mean values can be measured with Cohen's d, which is defined as

$$d = \frac{mean(X_1) - mean(X_2)}{s}$$

where

$$s = \sqrt{\frac{(n_1 - 1) \cdot sd(X_1)^2 + (n_2 - 1) \cdot sd(X_2)^2}{n_1 + n_2 - 2}}.$$

While s looks complicated, this is nothing else but *pooled standard deviation* of the two samples, i.e., the weighted mean of the standard deviations of the two samples, where the weights are determined by the number of samples. Thus, Cohen's d measure the ratio of the difference between the mean values to the standard deviation and the meaning of $d = 1$ is that the difference of sample means is one standard deviation. According to Cohen and Sawilowsky, the following table can be used as a guide to interpret the *magnitude* of the effect size.

Cohen's d	Effect size
$\|d\| \geq 0.01$	Very small
$\|d\| \geq 0.2$	Small
$\|d\| \geq 0.5$	Medium
$\|d\| \geq 0.8$	Large
$\|d\| \geq 1.2$	Very large
$\|d\| \geq 2.0$	Huge

We note that this table was designed for use within the social sciences. Regardless, these values are broadly used, independent of the domain.

In our example, we have the following effect size.

```
Effect size (Cohen's d): -0.393 - small effect size
```

Thus, we have a small effect size. This is in line with the histogram of the data: we see that there may be a difference between the two samples, but the difference is relatively small. This is confirmed by Cohen's d.

11.4 Confidence Intervals

Finally, we may wonder how trustworthy the estimation of mean values is. This depends on two factors: the number of instances in a sample and the variability of the sample. The following example makes the intuition behind this clear. Imagine you want to know the average age of your customers. Your estimation obviously gets more accurate, the more customers you ask for their age. Moreover, imagine the first customer answer 18, the second 75, the third 22, the you observe 44, etc. This means there is obviously a large variability in the customer age. To counter this, you must ask more customers.

In the following, we use an example that directly relates to machine learning. Imagine you have twenty different data sources (e.g., data from twenty stores) to train a classification model, e.g., for customer churn. You use the data from five stores for training, and the data from the remaining 15 stores to evaluate the quality of the model. This means you get 15 performance estimates. Now assume that you observe a normal distribution of the accuracy mean value of 0.83 and a standard deviation of 0.13. The question is how accurate this mean accuracy is. In other words: what could you expect, if you apply your model to fifteen other scores?

The *confidence interval* allows this kind of estimation. It is defined as follows.

> *Definition: Confidence Interval*
>
> A $C\%$ confidence interval θ for some parameter p is an interval that is expected with probability $C\%$ to contain p.

Thus, there can be a confidence interval for any statistical marker p. The idea is that the actual value of p is somewhere within the interval. In our case, p is the mean value of a normal distribution. C is called the confidence level. For example, if $C = 0.95$, we expect that the there is a 95% chance that future estimations of the mean value would be within the confidence interval.

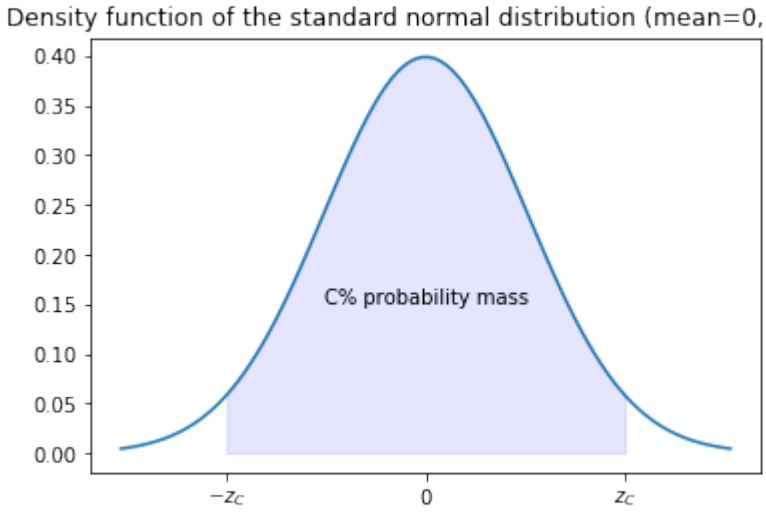
This confidence interval of the mean of a normally distributed sample $X = \{x_1, ..., x_n\}$ can be calculated as

$$\theta = [mean(X) - z_C \frac{sd(X)}{\sqrt{n}}, mean(X) + z_C \frac{sd(X)}{\sqrt{n}}]$$

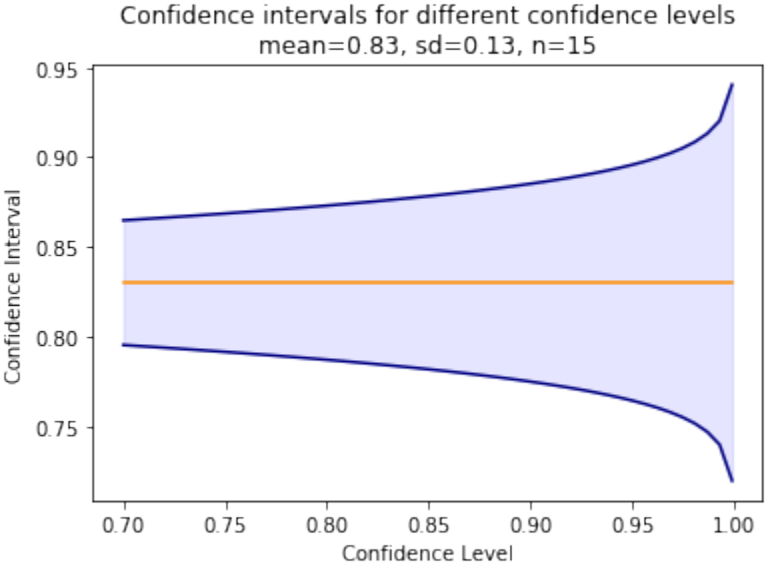
where z_C is the so-called *z-value* for the confidence level. We see that this Definition matches the intuition: the interval gets smaller, with more samples and/or a smaller standard deviation. In practice, z_C is often just looked up in a table.

C	z_C
90%	1.645
95%	1.96
99%	2.58
99.5%	2.807
99.9%	3.291

The origin of the z-value is the standard normal distribution. The z-value is choosen such that the area under the probability density function is C.



For our example, the confidence interval of our example looks as follows for different confidence levels.



Same as the p-value, the confidence interval is often misinterpreted. The correct interpretation of a $C\%$ confidence interval is, e.g., one of the following.

- There is a $C\%$ chance that the results of future replications of the same experiment fall into θ.
- There is no statistically significant difference of the estimated parameter for all values within θ with a $C\%$ confidence.
- There is a $1 - C\%$ probability of the observed data, if the true value for the estimated parameter is outside of θ.

Common misinterpretations of the confidence interval that should absolutely be avoided are, e.g., the following.

- The true value of p lies with $C\%$ probability in θ. Same as hypothesis tests, confidence intervals are a statement about the observed data.
- $C\%$ of the observe data is in θ. This mixes up the confidence interval with the quantiles.

11.5 Good Statistical Reporting

While many publications still rely only on reporting the p-value and simply state that a result is significant, good statistical reporting goes beyond this and also considers additional aspects, such as effect sizes and confidence intervals. Moreover, all statistical analysis should be planned beforehand and not be conducted in an exploratory fashion, as this leads to p-hacking. Therefore, many scientific fields are turning towards *registered reports*, where the planned statistical analysis is described in detail before conducting the study. This in an effective measure to prevent exploratory statistics, which lead to false positive findings.

Moreover, the statistics we discussed here is *frequentist statistics*. Good reporting may also use a completely different approach, i.e., *Bayesian statistics*. Bayesian statistics is based on Bayes theorem and evaluates data through the estimation of the *degree of belief* in an outcome. Scientists in favor of Bayesian approaches argue that the results are often easier to interpret than frequentist statistics and better aligned with the actual research questions.

CHAPTER

TWELVE

BIG DATA PROCESSING

12.1 Distributed Computing

As we know from Chapter 1, Big Data is data whose volume, velocity, and variety requires *innovative forms of information processing*. In this chapter, we want to discuss in greater detail why this is the case and how Big Data can be processed.

The foundation of any large computational effort is *parallelism*. There is a famous quote from computer science pioneer Grace Hopper: "*In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a larger ox. We should be trying for bigger computers, but for more systems of computers*. In other words, large tasks can only be solved by pooling resources. There are three general methods for the parallelization of computational tasks.

12.1.1 Parallel Programming Models

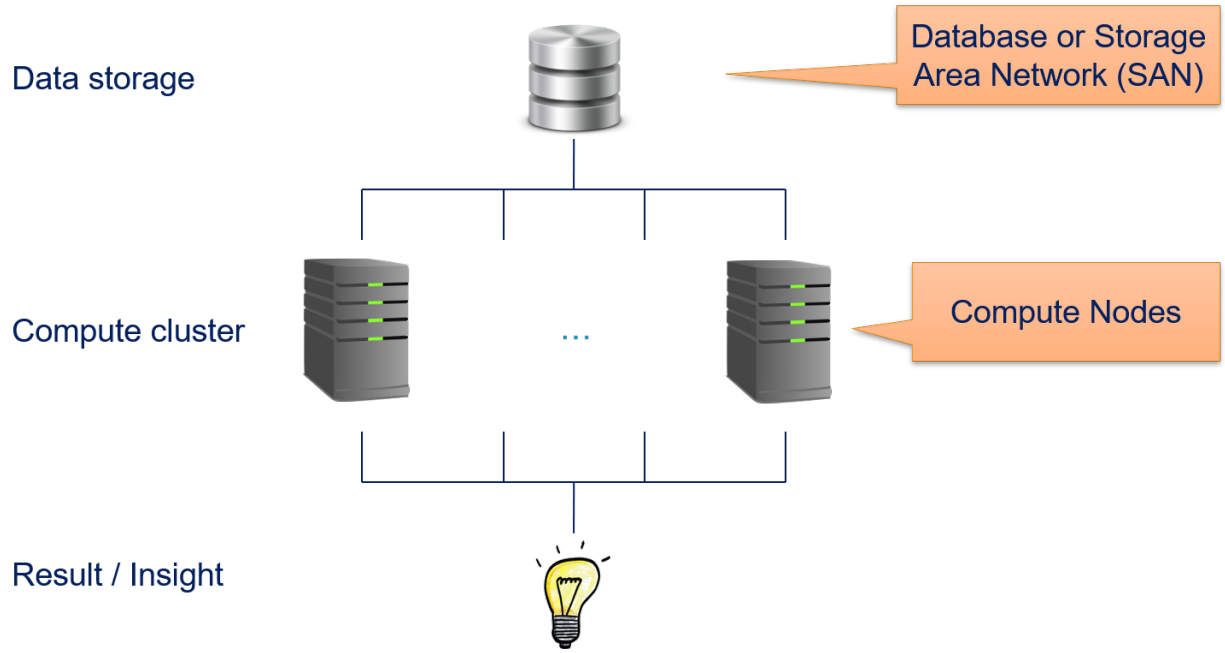
First approach *message passing* tasks are executed independently in isolated environments. Whenever these tasks want to communicate, they send messages to each other. This way, the tasks can exchange data between each other, e.g., because the data is required by different parts of the computation. This communication can be done locally on one physical machine, by using the provided functions by the operating system, or remotely in a distributed environment by communicating via the network.

The second approach is *shared memory*. In this case, the computational tasks are not performed in isolated environments, but share a common address space in the memory, i.e., they can read and write the same variables. Interactions between the tasks happens by updating the values of variables in the shared memory. Sharing memory within a single physical machine is directly supported by the operating system, and may even be a property of the model for parallelization (threads share the same memory, processes not). Sharing memory across different physical machines is also possible, e.g., via network attached storage of other networking solutions, but usually has some communication overhead.

The third approach is *data parallelism*. Similar to message passing, tasks are executed independently in isolated environments. The difference to message passing is that the tasks do not need to communicate with each other, because the solution of the computational tasks does not require intermediary results of other tasks. Thus, the application of data parallelism is limited to problems where this strong decoupling of tasks is possible. Such problems are also called *embarrassingly parallel*.

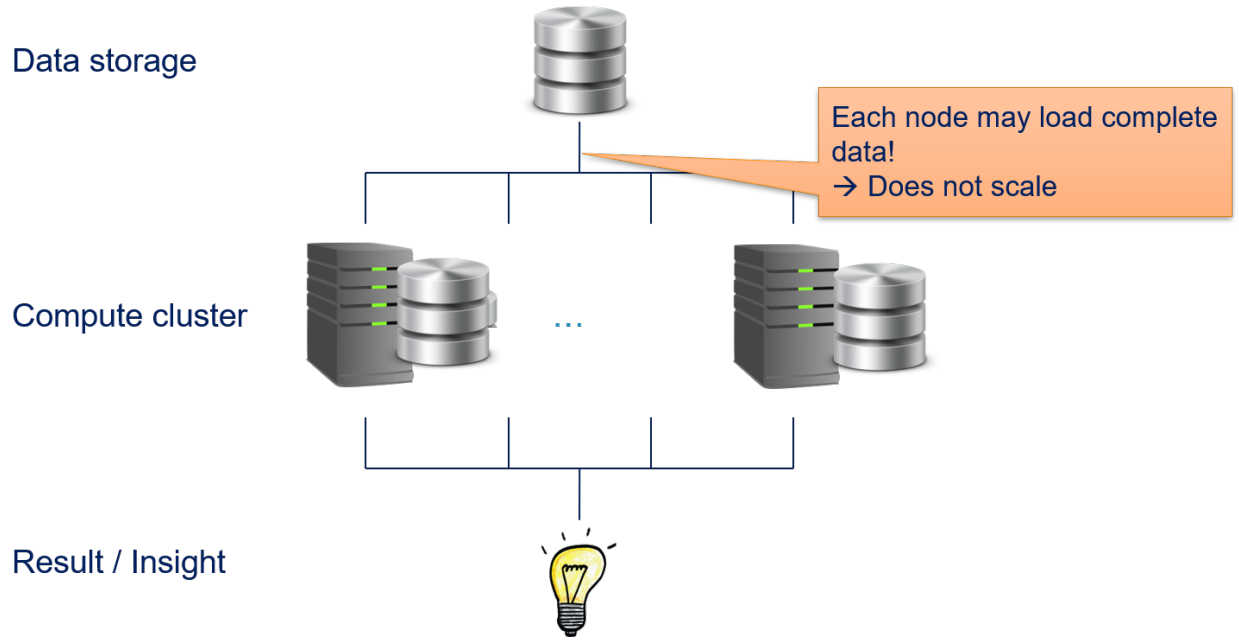
12.1.2 Distributed Computing for Data Analysis

Since Big Data is to large too compute or store on single physical machines, we need a distributed environment for computations that involve Big Data. Before computational centers started to account for Big Data, the architecture of such a *compute cluster* was similar to the outline below.



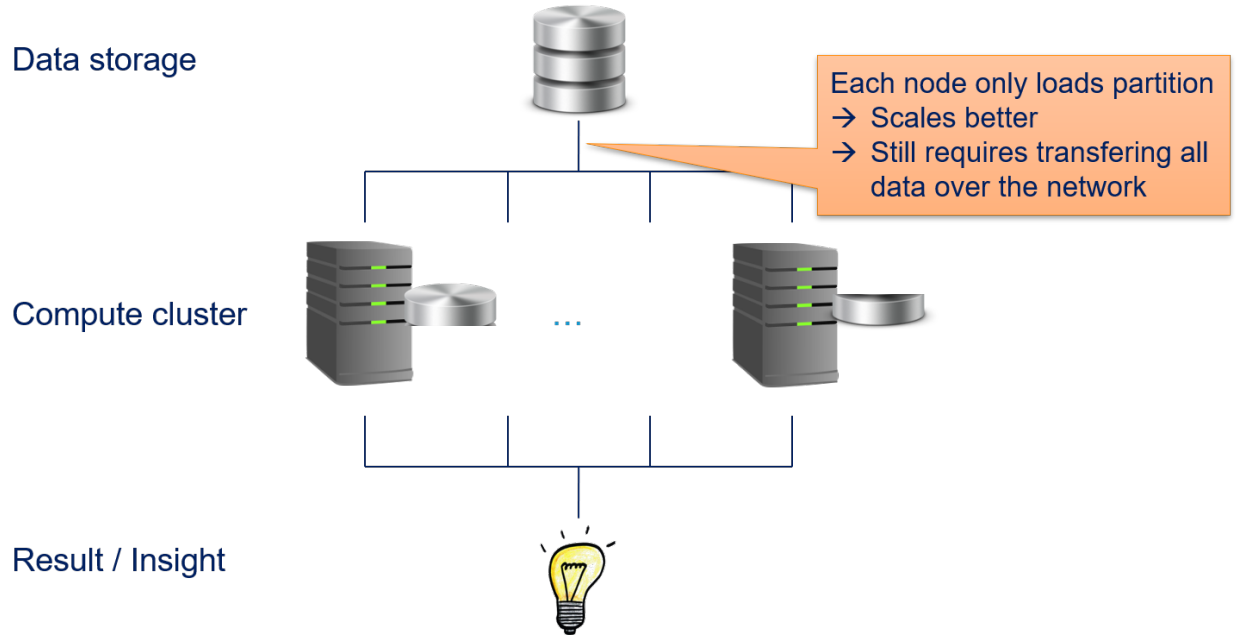
There is a layer for data storage and a layer for computations. Both are using different *nodes* in the compute cluster. Each node is a physical machine. Data storage nodes must provide fast storage (latency, throughput, or both), but do not require much computational power. This is usually implemented through database or a *storage area network* (SAN). Vice versa, compute nodes must provide the computational power through CPUs (and possibly GPUs) and a sufficient amount of memory, local storage is less important and often only used for caching and the installation of software. A user of such a system submits jobs to a job queue to gain insights. For the analysis of data, this means that the data is stored in the database or SAN and then accessed by the compute nodes to generate the desired results of the analysis, from which the data scientists can get insights.

All three parallelization modes we discussed above can be implemented in such a traditional distributed compute cluster. However, none of these approaches is suitable for big data applications in such a compute cluster. Message passing and shared memory have the biggest scalability problems.



Since it is unclear which parts of the data are required by the different parallel tasks, it is possibly that every compute node must load all data. While this is not a problem for small data sets, this does not scale with large data sets. Imagine that Terabytes, or even Petabytes of data would have to be copied regularly over the network. The transfer of the data would block the execution of the analysis and the compute nodes would be mostly idle, waiting for data. This does not even account for additional network traffic due to the communication between the tasks.

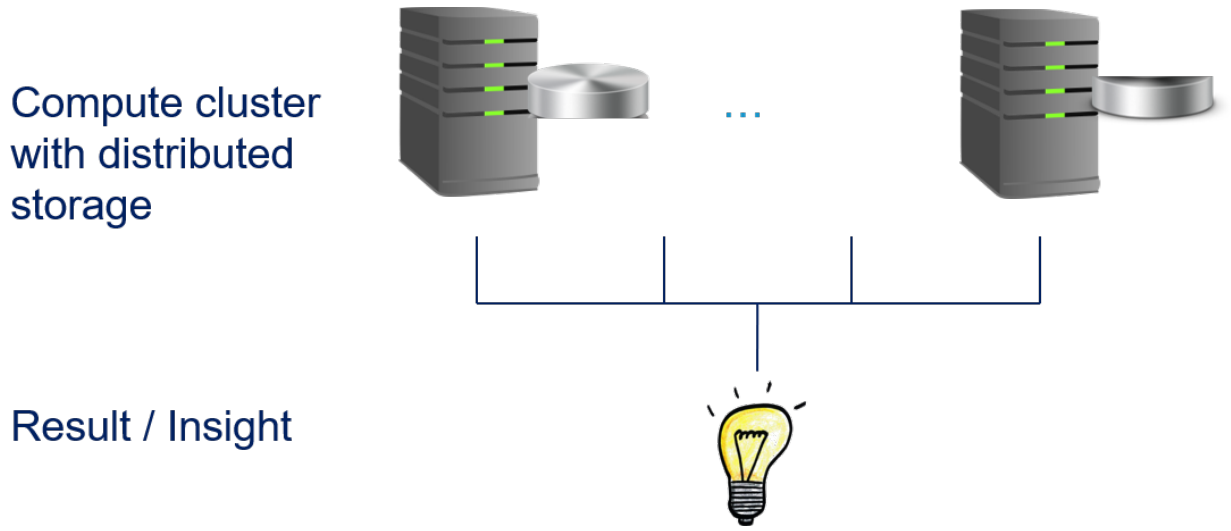
Data parallelization fares a bit better, but also does not scale.



The advantage of message passing and shared memory is that only parts of the data must be copied to each compute node. While this decreases the stress on the network, all data must still be transferred over network. Thus, data parallelization can handle larger amounts of data than message passing and shared memory, at some point the amount of data becomes too large for the transfer via the network.

12.1.3 Data Locality

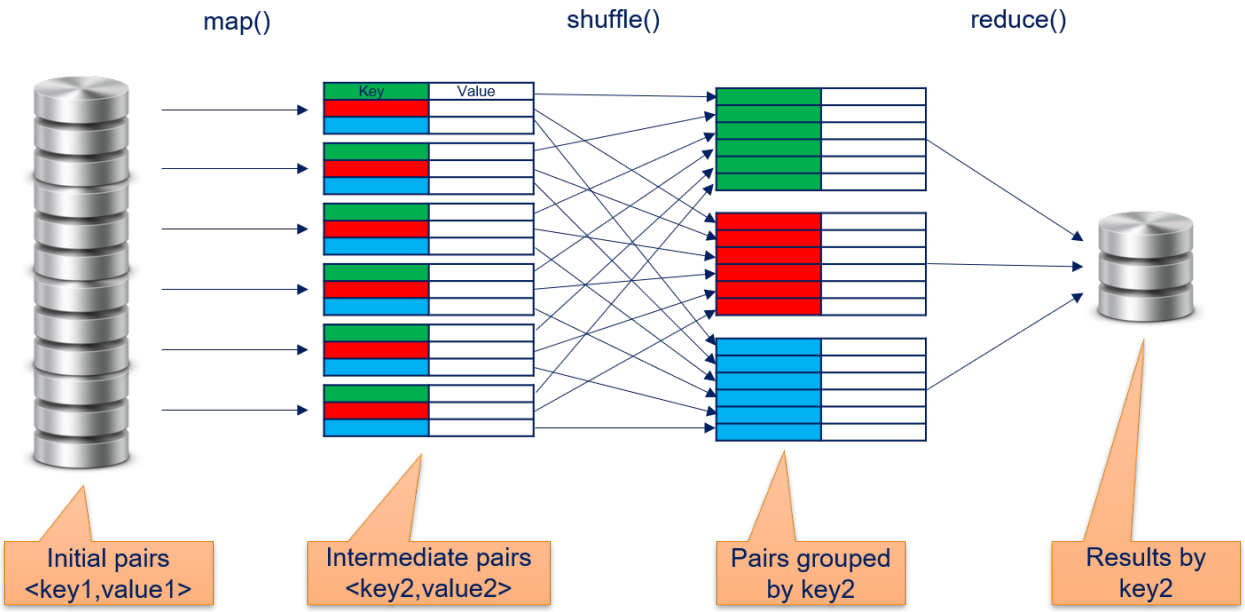
We see that there is a fundamental problem with traditional distributed computing for big data, which is why we need the *innovative forms of information processing*. The solution is actually quite simple: if the problem is that we cannot copy our data over the network, we must change our architecture such that avoid that. The straightforward way to achieve this is to break the separation of the storage layer from the compute layer: all nodes both store data and can perform computations on that data.



In the following, we explain how this is implemented in practice. We discuss the MapReduce programming model that became the de facto standard for Big Data applications. Then, we show Apache Hadoop and Apache Spark to demonstrate how the distributed computing with Big Data is implemented.

12.2 MapReduce

The MapReduce paradigm for data the data parallelization to enable Big Data processing was published by Google in 2004. The general idea is to describe computations using two different kinds of functions: *map* functions and *reduce* functions. Both functions work with *key-value pair*. Map functions implement the embarrassingly parallel part of algorithms, reduce functions aggregate the results. The concept of map functions and reduce functions is not unique to MapReduce, but a general concept that can be found in many functional programming languages. To enable Big Data, MapReduce introduces a third function, the *shuffle*. The only task of the *shuffle* is to arrange intermediate results, i.e., to facilitate the communication between the map and reduce functions. The following figure gives an overview of the dataflow of MapReduce.



12.2.1 map()

The map function gets initial key-value pairs. These are, e.g., read from the distributed storage or the result of a prior computation using MapReduce. The map function than performs a computation on a *single* key-value pair and stores the results in new key-value pairs. Because the map function only gets a single key-value pair as input, data parallelization is trivial: theoretically, the map function could run in parallel for all key-value pairs without any problem. The map function is defined as

$$map(f_{map}, < key1, value1 >) \rightarrow list(< key2, value2 >)$$

where f_{map} is a *user-defined function* (UDF) defined by the user of the MapReduce framework. The UDF defines the computation, i.e., how the input key-value pair is transformed into the list of output key-value pairs. Depending on the UDF f_{map}, the input keys and output keys could be same or different. While the general concept of MapReduce does not have any restrictions on the type and values of the keys, implementations of MapReduce may restrict this. For example, in the initial implementation of MapReduce by Google, all keys and values were strings and users of MapReduce were expected to convert the types within the map and reduce functions, if required.

12.2.2 shuffle()

The key-value pairs computed by the map function are organized by the shuffle function, such that the data is grouped by the key. These are then organized by the shuffle and grouped by their keys. Thus, we have

$$shuffle(list < key2, value2 >) \rightarrow list(< key2, list(value2) >),$$

i.e., a list of values per key. Often, these data from shuffle is sorted by key, because this can sometimes improve the efficiency of subsequent tasks. The shuffling is often invisible to the user performed in the background by the MapReduce framework.

12.2.3 reduce()

The reduce function operates on all values for a given key and aggregates the data into a single result per key. The reduce function is defined as

$$reduce(f_{reduce}, < key2, list(value2) >) \rightarrow value3$$

where f_{reduce} is a UDF. The UDF f_{reduce} performs the reduction to a single value for one key and gets as input the key and the related list of values. Similar as for the map function, there is no restriction on the type or the values that are generated. Depending on the task, the output could, e.g., be key value pairs, integers, or textual data.

12.2.4 Word Count with MapReduce

The concept of MapReduce is relatively abstract, unless you are used to functional programming. How MapReduce works becomes clearer with an example. The "Hello World" of MapReduce is the word count, i.e., using MapReduce to count how often each word occurs in a text. This example is both practically relevant, e.g., to create a bag-of-words, and well suited to demonstrate how MapReduce works.

We use the following text as example:

```
What is your name?
The name is Bond, James Bond.
```

Our data is stored in a text file with one line per sentence. Our initial keys are the line numbers, our initial values the text in the lines. Thus, we start with these key-value pairs.

```
<line1, "What is your name?">
<line2, "The name is Bond, James Bond.">
```

The map function is defined such that it emits the pair for each word in the input. When we apply this to our input, we get the following list of key-value pairs.

```
<"what", 1>
<"is", 1>
<"your", 1>
<"name", 1>
<"the", 1>
<"name", 1>
<"is", 1>
<"bond", 1>
<"james", 1>
<"bond", 1>
```

The shuffle then groups the values by their keys, such that all values for the same key are in a list.

```
<"bond", list(1, 1)>
<"is", list(1, 1)>
<"james", list(1)>
<"name", list(1, 1)>
<"the", list(1)>
<"what", list(1)>
<"your", list(1)>
```

As reduce function, we output one line for each key. The lines contain the current key and the sum of the values of that key.

```
bond 2
is 2
james 1
name 2
the 1
what 1
your 1
```

12.2.5 Parallelization

The design of MapReduce enables parallelization for every step of the computational process. The input can be read in chunks to parallelize the creation of the initial key-value pairs. For example, we could have multiple text files, each with 1000 lines that could be processed in parallel. The parallelism is limited by the throughput of the storage and makes more sense, if the data is distributed across multiple physical machines.

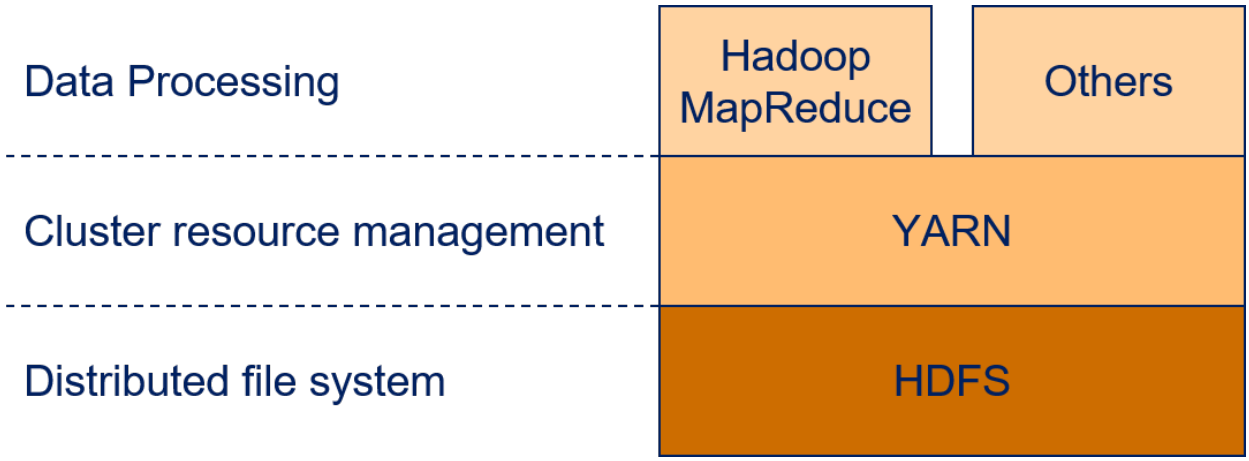
map() can be applied to each key-value pair independently and the potential for parallelism is only limited by the amount of data.

shuffle() can start as soon as the first key-value pair is processed by the map function. This reduces the waiting times, such that the shuffling is often finished directly after the last computations for map are finished.

reduce() can run in parallel for different keys. Thus, the parallelism is only limited by the number of unique keys created by map(). Moreover, reduce() can already start, once all results for a key are available. This is where sorting by shuffle can help. If the results passed to reduce() are sorted, reduce can start processing for a key, once it sees results for the next key.

12.3 Apache Hadoop

Apache Hadoop is an open source implementation of MapReduce. For many years, Hadoop was the standard solution for any MapReduce application and Hadoop is still relevant for many applications. All major cloud providers offer Hadoop clusters in their portfolio. Hadoop 2.0 implements MapReduce in an architecture with three layers.



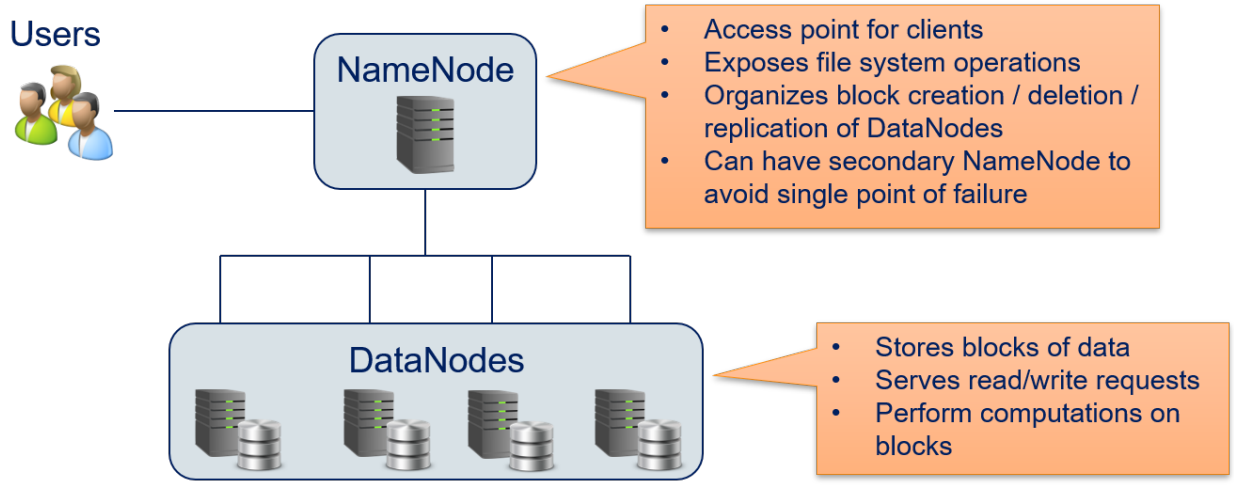
The lowest layer is the *Hadoop Distributed File System* (HDFS) that is in charge of the data management. *Yet Another Resource Negotiator* (YARN) is running on top of the file system. YARN manages the use of computational resources within a Hadoop cluster. Applications for data processing that want to use the Hadoop cluster are running on top of YARN. For example, such applications can be written with the Hadoop implementation of MapReduce. However, due to the success of the HDFS and YARN, there are also other technologies that can be used for data processing, e.g., *Apache Spark*, which we discuss below.

12.3.1 HDFS

The HDFS is the core of Hadoop. All data that should be analyzed is stored in the HDFS. HDFS was designed with the goal to enable Big Data processing, which is why HDFS behave quite differently from other file systems that we regularly use like NTFS, ext3, or xfs.

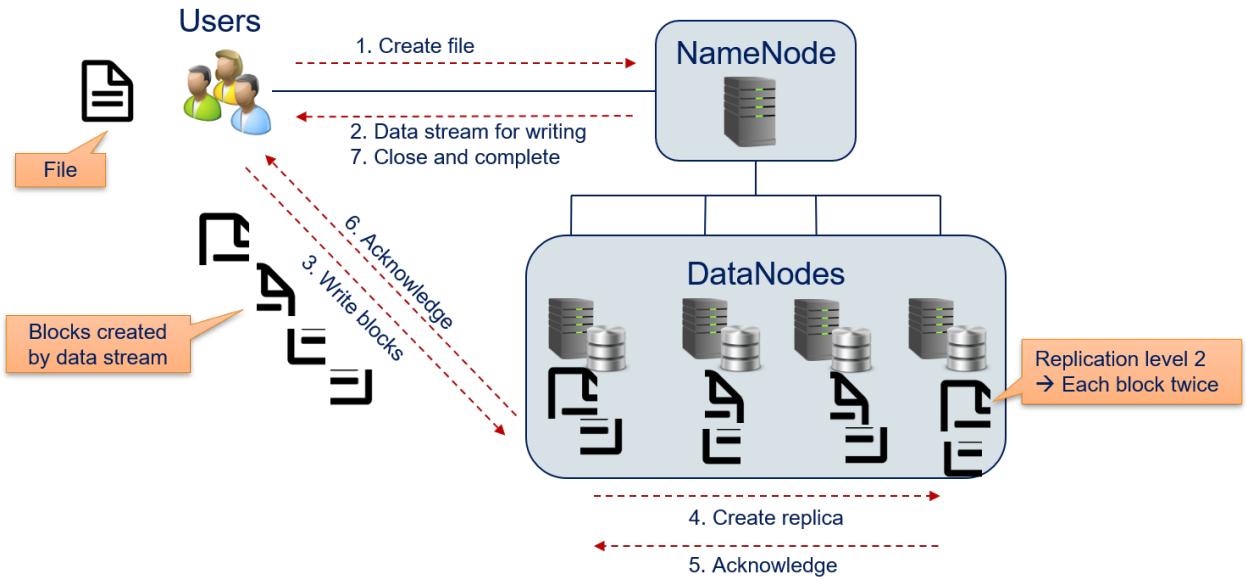
- HDFS favors high throughput at the cost of low latency. This means that loading and storing large amounts of data is fast, but you may have to wait some time before the operation starts.
- HDFS supports extremely large files. The file size is only limited by the amount of *distributed* storage, i.e., files can be large than the storage available at a single node.
- HDFS is designed to support data local computations and minimize the data that needs to be send around in a cluster for file operations.
- Since outages of single nodes in a large compute cluster is not a rare event, but rather a part of the daily work, HDFS is designed to be resilient against hardware failures, such that there is no loss of data and no interruption of service.

In principle, HDFS uses a master/worker paradigm with a *NameNode* that manages *DataNodes*.



Clients access the HDFS via the NameNode. All file system operations, such as the creating, deletion, copying of a file is performed by requesting this at the NameNode. Whenever a file is created, it is split into smaller blocks. The NameNode organizes the creation, deletion, and replication of *blocks* on DataNodes. Replication means that each block is not just stored on a single DataNode, but on multiple DataNodes. This ensures that no data is lost, if a data node is lost. To avoid that the HDFS is not available if the NameNode crashes, there can also be a secondary NameNode. This avoids that the NameNode is a single point of failure. If there is a problem with the primary NameNode, the secondary NameNode can take over without loss of service.

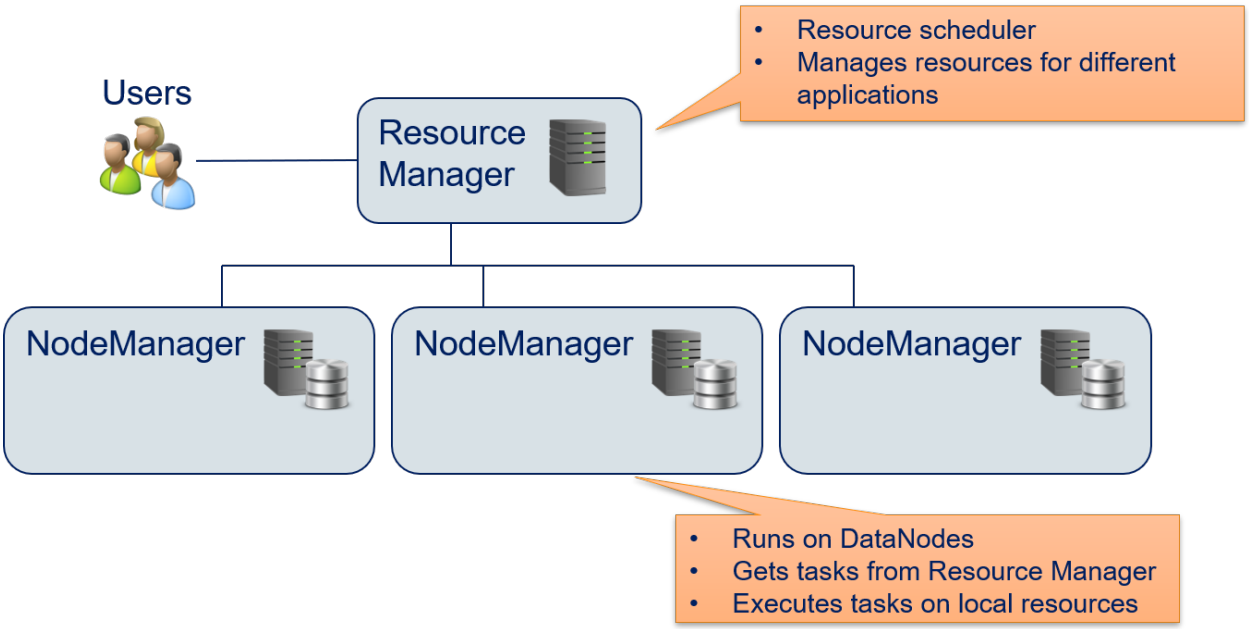
Another important aspect of the HDFS is that while users access the HDFS via the NameNode, the actual data is never sent via the NameNode to the DataNodes, but directly from the users to the DataNodes. The following figure shows how a file in HDFS is created by a user.



1. The user contacts the NameNode with the request to create a new file.
2. The name node responds with a data stream that can be used for writing the file. From the user's perspective, this is a normal file stream that would be used for local data access (e.g., `FileInputStream` in Java, `ifstream` in C++, `open` in Python).
3. The user writes the contents of the file to the file stream. The NameNode configured the file stream with the information how the blocks should look like and where the data should be sent. The data is directly sent block-wise to one of the DataNodes.
4. The DataNode that receives the block does not necessarily store the block itself. Instead, the block may be forwarded to other DataNodes for storage. Each block is stored at different nodes and the number of these nodes is defined by the *replication level*. Moreover, the HDFS ensures that blocks are evenly distributed among the data nodes, i.e., a DataNode only stores multiple blocks of a file, if the block cannot be stored at another DataNode. Ideally, all DataNodes store the same number of blocks of a file.
5. When a DataNode stores a block, this is acknowledged to the DataNode that receives the data from the user.
6. Once the DataNode received acknowledgements for all replications of a block, the DataNode acknowledges to the user that the block is stored. The user can then start sending the data for the next block (go to step 3), until all blocks are written. The user does not observe this behavior directly, because this is automatically handled by the file stream.
7. The user informs the NameNode that the writing finished and closes the file stream.

12.3.2 YARN

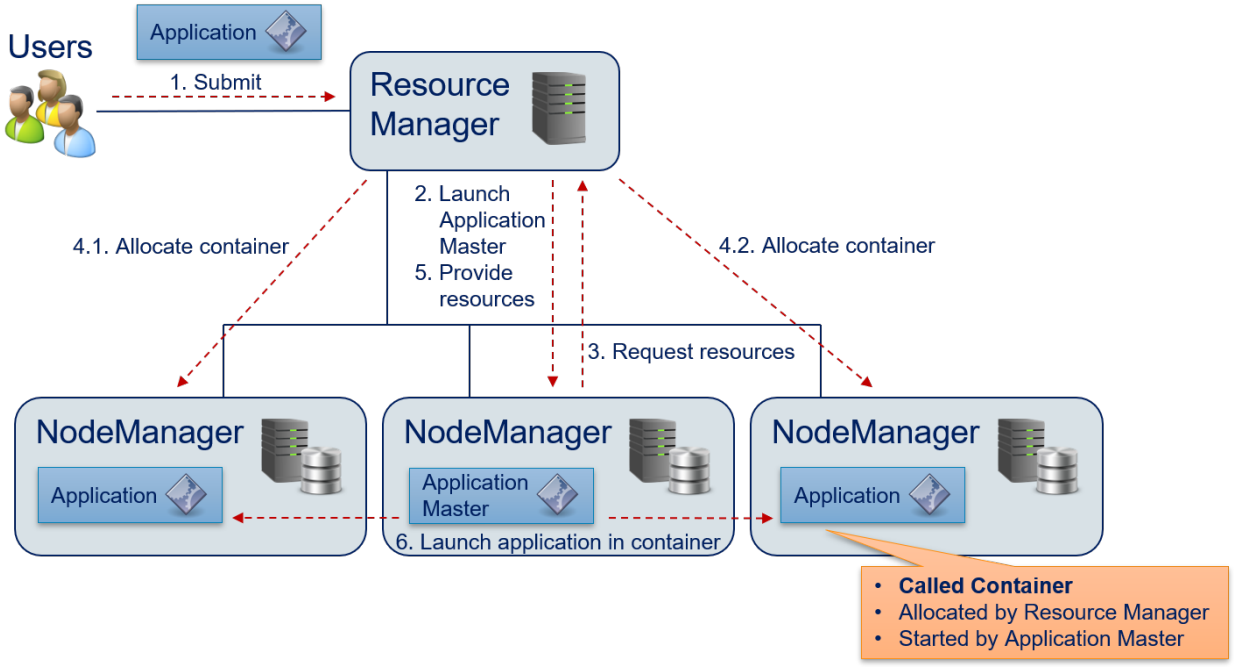
The second core component of Hadoop is YARN, which is manager for computational resources that is designed to enable distributed Big Data computations with data stored in the HDFS. Same as the HDFS, YARN has a master/worker paradigm.



The *Resource Manager* is the scheduler that provides resources for applications to enable the computations. The *NodeManagers* are applications running on the DataNodes. The NodeManager execute tasks on the local resources. This way, each DataNode can serve as compute node. The computational tasks are allocated such that they are executed by the NodeManagers running on the DataNodes where the required data is stored. Thus, the combination of DataNodes and NodeManagers running on the same physical machine and the Resource Manager that is aware where the data is stored in the HDFS enables data local computations. The computational tasks are, e.g., executed using MapReduce. However, YARN can in principle run any kind of computational task on the NodeManagers and is not limited to MapReduce.

The Resource Manager should schedule the computational task such that the resources (CPU cores, memory) are spent efficiently. This means that NodeManagers should ideally only execute one task at a time, to prevent overutilization. Underutilization should also be avoided: if there are jobs waiting for their execution and at the same time, there are idle NodeManagers that could conduct the jobs, the jobs should get these resources. This means that the Resource Manager should must be able to provide resources to multiple jobs, possibly of multiple users, at the same time.

The following figure shows how YARN executes application in a distributed manner and schedules resources.



1. The user submits an application to the Resource Manager. The Resource Manager queues this application until there are sufficient capacities in the compute cluster to start the execution.
2. The Resource Manager allocates a *container* on one of the NodeManagers and launches the *application master*. The application master is not the application itself, but rather a generic program that knows how to execute the application, e.g., which resources are required and which tasks must be executed.
3. The application master requests the resources required for computations from the Resource Manager.
4. The node manager allocates the required containers. In the example, these are two containers running on the NodeManagers on the left and right side.
5. The Resource Manager informs the application master that the required resources are allocated. This information includes the data required to access the containers.
6. The application master executes (parts of) the application in the containers. The application managers may configure the environment through environment variables. The applications are then using local resources of the host where the NodeManager is running, e.g., binary or data from the HDFS.

Once the application is finished, all containers are destroyed. The results only accessible via the HDFS. The resource requests (step 3) may use the following information to specify the required resources:

- The required number of containers.
- CPU cores and memory that are required per container.
- The priority of the request. This priority is local to the application and not global. This means that the priority only influences which of the resources a specific application requests it gets first. A higher priority does not give any advantages in scheduling with respect to other applications that are also requesting resources from the Resource Manager.
- It is also possibly to directly specify the name of the desired computational resources. This could either be a specific host of NameNode, but also a more generic property of the topography of the compute cluster, e.g., in which rack the host should be located.

12.3.3 MapReduce with Hadoop

Hadoop provides a MapReduce implementation that uses YARN for the execution. In such a MapReduce application, users define sequences of map()/reduce() tasks to solve their problem. The execution is driven by the MRAppMaster Java application. This application specifies a YARN application master that manages the execution of tasks, in this case of the map() and reduce() tasks. Users specify the MapReduce applications through a Java application. Hadoop also has a streaming mode for job execution, which we discuss below.

The tasks of MapReduce applications are specified through subclassing. Subclasses of the `Mapper` class define map() functions, subclasses of the `Reducer` class define `reduce()`. The code below specifies the map() function for the word count example. Please note that we omit all boilerplate code from the code samples, e.g., import statements. The complete example can be fount in the official Hadoop documentation.

```
public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    @Override
    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
        // text into tokens
        StringTokenizer itr = new StringTokenizer(value.toString().toLowerCase());
        while (its.hasMoreTokens()) {
            // add an output pair <word, 1> for each token
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}
```

The `TokenizerMapper` class extends to generic class `Mapper` with four parameters of types `Object`, `Text`, `Text`, and `IntWritable`. The first two parameters specify the types of the key and value of the input key-value pairs of the map() function, which means we have keys of type `Object` and values of type `Text`. the last two parameters specify the type of output key-value pairs, which means we have keys of type `Text` with values of type `IntWritable`. `Text` and `IntWritable` are data types provided by Hadoop that are similar to their Java counterparts `String` and `Integer`. The key differences between the Hadoop types and the standard Java types are that of the Hadoop types are *mutable*, i.e., the values of objects can be modified, and the Hadoop types are optimized for serialization to improve the efficiency of exchanging key-value pairs between different map() and reduce() tasks.

The class has two attributes `one` and `word`, which are used to generate the output keys and values. These advantage of having these as attributes is that they are not initialized with every call of map(), which improves the efficiency. Finally, we have map() function, defines how the input key values are mapped to the output pairs. In addition to the input key and value, map() gets the `context`. This context specifies the Hadoop execution environment and contains, e.g., values of environment variables. Moreover, the context receives the output of the map() function through the `context.write()` method. Thus, the map() function does not return values, but continuously writes results for which the computation finished to the context. The context contains the shuffle() function and can immediately start shuffling the output key-value pair, once it was written. Thus, shuffling can already start before the first map() function finished.

The code below shows the reduce() function for the word count.

```
public static class IntSumReader extends Reducer<Text, IntWritable, Text, IntWritable>
↪ {
    private IntWritable result = new IntWritable();

    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context
```

(continues on next page)

(continued from previous page)

```
                        ) throws IOException, InterruptedException {
        // calculate sum of word counts
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        // write result
        context.write(key, result);
    }
}
```

The `IntSumReader` class extends the generic class `Reducer`. Same as for the `TokenizerMapper`, the parameters describe the types of the input, respectively out, the attribute `result` is used for efficiency, and the result is written to the context. The only notable difference between the classes is that the reduce function gets an `Iterable` of values and not a single value, i.e., all values for the key.

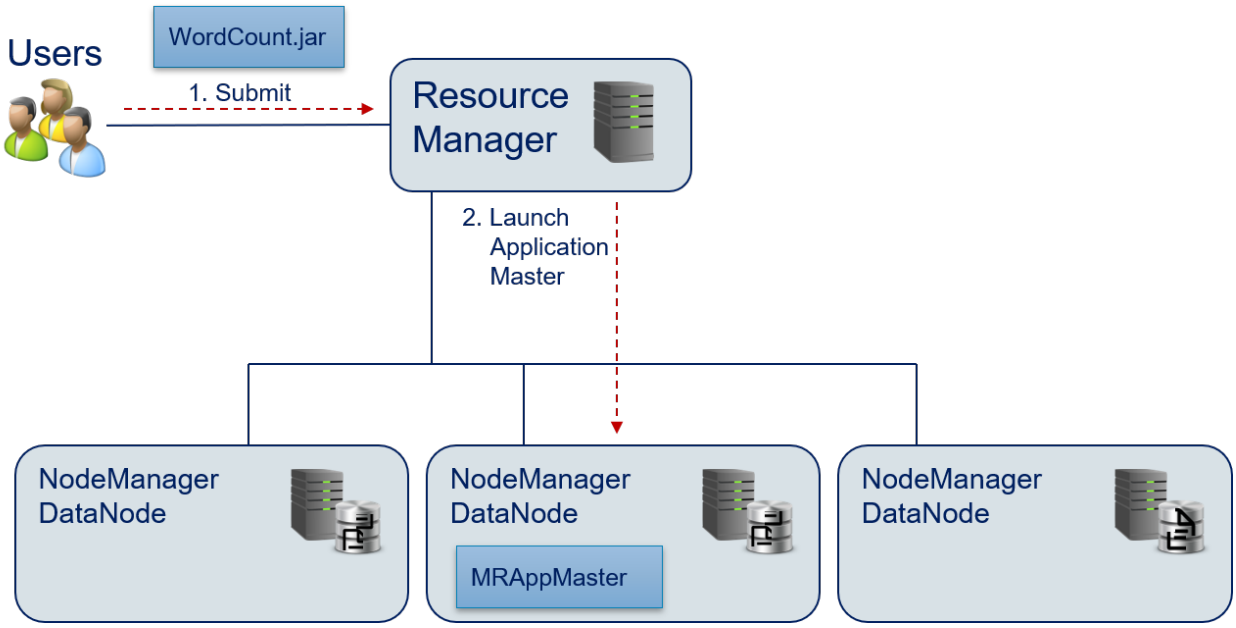
Finally, we need to use these classes in a MapReduce application. The code for the application is below.

```
public class WordCount {
    public static void main(String[] args) throws Exception {
        // Hadoop configuration
        Configuration conf = new Configuration();

        // Create a Job with the name "word count"
        Job job = Job.getInstance(conf, "word count");
        job.setJahrByClass(WordCount.class);

        // set mapper, reducer, and output types
        job.setMapperClass(TokenizerMapper.class);
        job.setReduczer(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        // specify input and output files
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // run job and wait for completion
        job.waitForCompletion(true);
    }
}
```

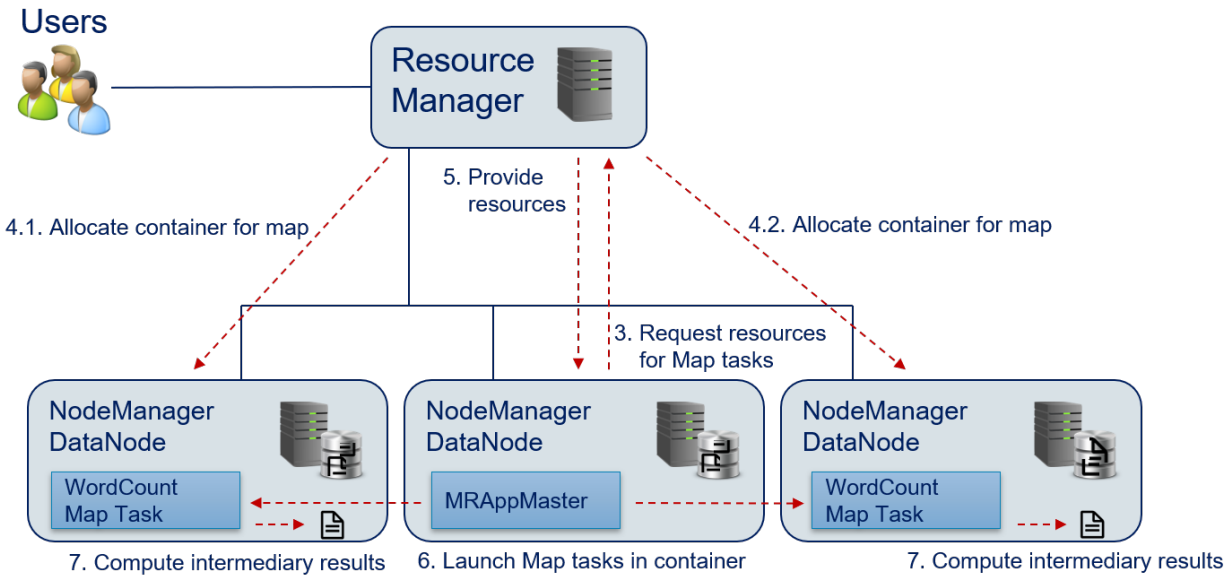
The application is a normal Java application with a main method and the code is more or less self-explanatory. First, we create the configuration of Hadoop application. This object contains, e.g., the context and MapReduce tasks. We use the configuration to create a new MapReduce job. Then, we configure the Job. We define in which class it is defined, which classes are used for the map() and reduce() steps, and what the types of the outputs are. Finally, we specify where the input data come from and where the output should be written, based on the command line arguments of the application. The final line starts the job and blocks further execution, until the job is finished, i.e., all map() and reduce() functions finished their work and the results are available in the file system.

12.3.4 Word Count with Hadoop

Since it is not intuitive how Hadoop/YARN orchestrate the execution of a MapReduce application, we now demonstrate this in detail step by step using the word count example.



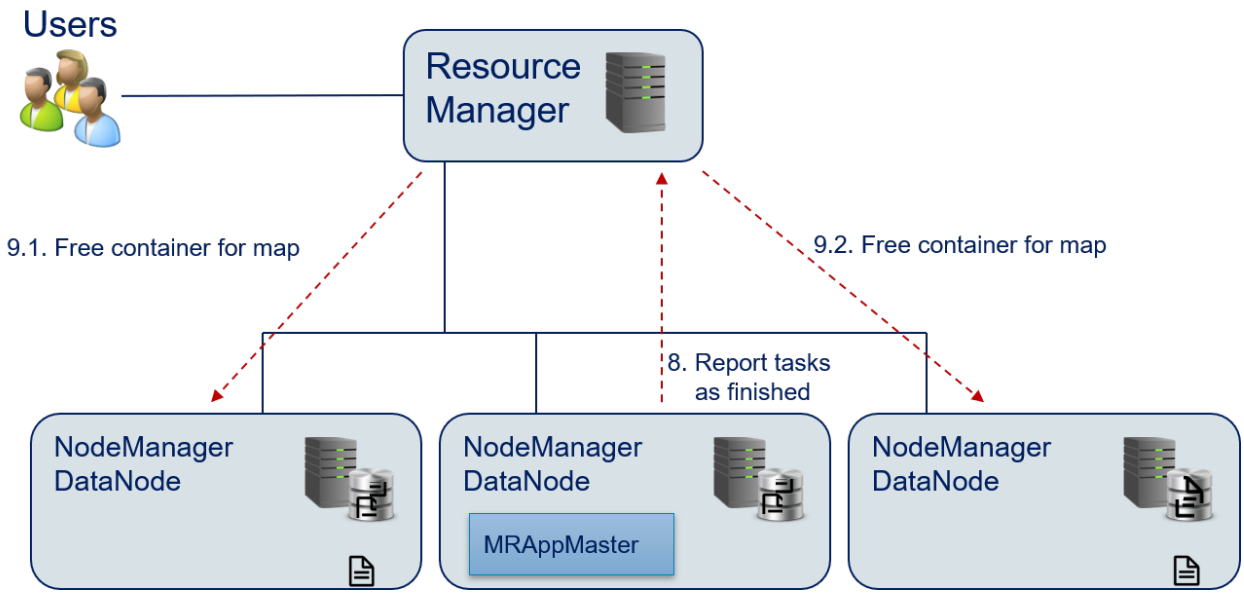
1. The user builds the jar-Archive of the MapReduce application and submits this application to the Resource Manager.
2. The Resource Manager launches the MRAppMaster application master to orchestrate the execution of the WordCount.jar.



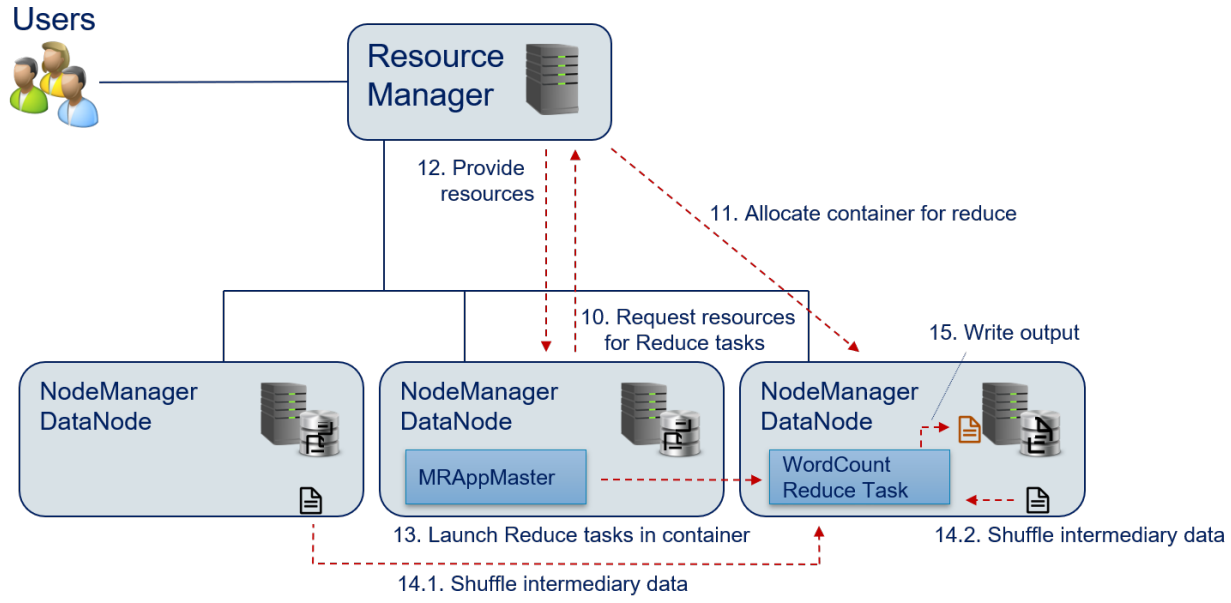
3. The MRAppMaster evaluates the configuration of the Hadoop application in the WordCount.jar and finds one jobs that consists of a map() and a reduce() function. The MRAppMaster requests the required resources for the map() function from the Resource Manager.
4. The Resource Manager provides the required compute resources. The resources are allocated on the DataNodes,

where blocks of the input data are stored, such that this data does not need to be transfered over the network.

5. The Resource Manager sends the information where the map() task can be executed to the MRAppMaster.
6. The MRAppMaster launches the map() function in the containers that were allocated. The required data is directly read from the HDFS.
7. The map() function is executed and the results are written by to intermediary results files in the HDFS.

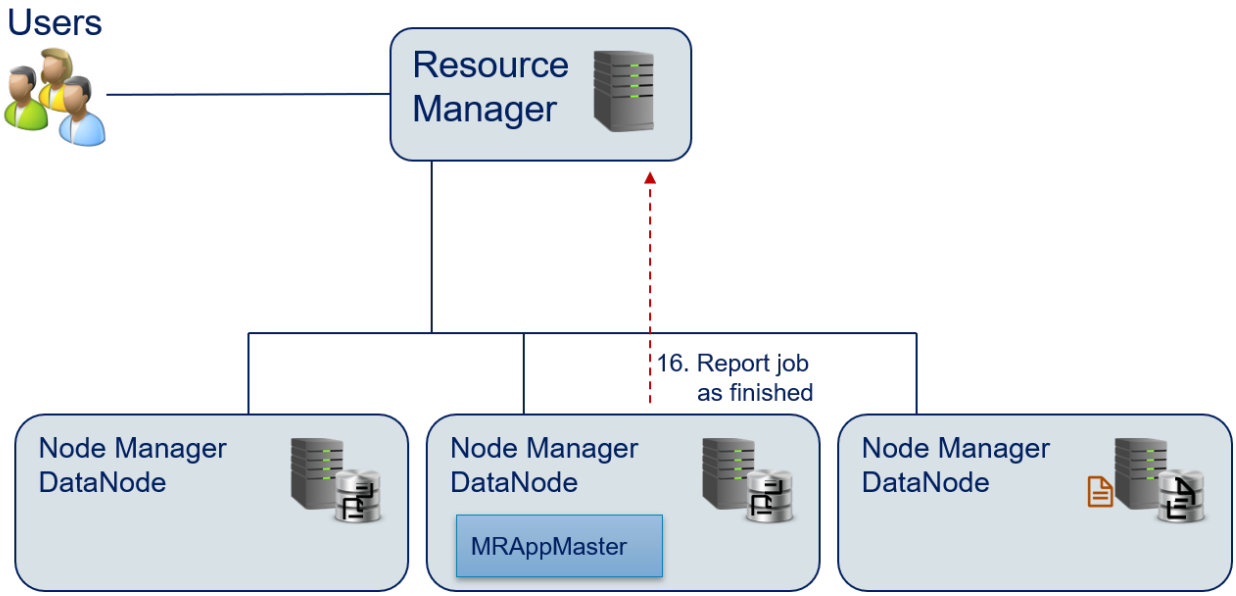


8. The MRAppMaster reports that the task is finished and the resources are no longer required to the Resource Manager.
9. The Resource Manager destroys the containers for the map() functions to free the resources.

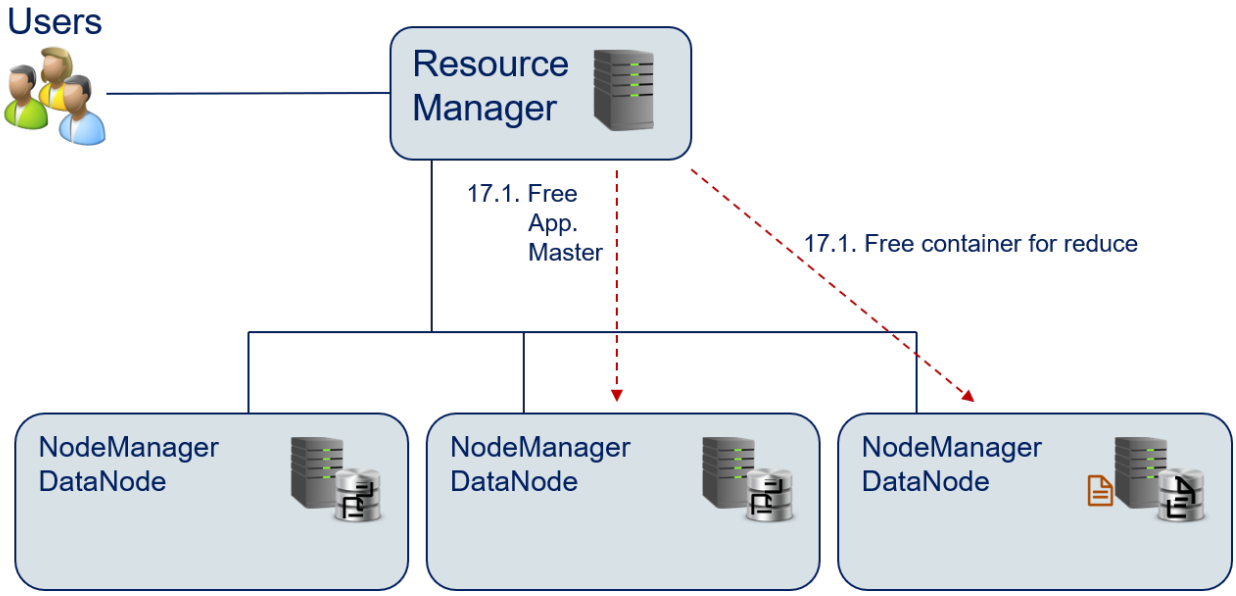


10. The MRAppMaster requests the required resources for the reduce() function. This requires only a single container, because the results are aggregated.
11. The Resource Manager allocates the required resources.

12. The Resource Manager sends the information where the map() task can be executed to the MRAppMaster.
13. The MRAppMaster launches the reduce() function in the allocated container.
14. The MRAppMaster triggers the shuffle() function to organize and group the intermediary data from the different nodes and provide the data to the reduce() function.
15. The reduce() function writes the output to the HDFS.



16. The MRAppMaster reports that the task and the execution of the application are finished to the Resource Manager.



17. The Resource Manager destroys the containers of the reduce() function and the MRAppMaster to free the resources.

12.3.5 Streaming Mode

Hadoop also provides a Java Application that can be used to Hadoop in the *streaming mode*. In the streaming mode, the standard input and standard output are used, similar to Linux pipes. The data from the HDFS is read and forwarded to an arbitrary application that processes the data from the standard input. The results of the computation are written to the standard output. For example, a python script for the map() function of the word count would look like this.

```
#!/usr/bin/env python
"""mapper.py"""

import sys

# read from standard input
for line in sys.stdin:
    # split line into words
    words = line.strip().split()
    # create output pairs
    for word in words:
        # print output pairs to standard output
        # key and value are separated by tab (standard for Hadoop)
        print('%s\t%s' % (word, 1))
```

Similarly, the reduce() function would also use the standard input and output.

```
#!/usr/bin/env python
"""reducer.py"""

from operator import itemgetter
import sys

# init current word and counter as not existing
current_word = None
current_count = 0
word = None

# read from standard input
for line in sys.stdin:
    # read output from mapper.py
    word, count = line.strip().split('\t', 1)
    count = int(count)

    # Hadoop shuffle sorts by key
    # -> all values with same key are next to each other
    if current_word==word:
        current_count += count
    else:
        if current_word:
            # write result to standard output
            print('%s\t%s' % (current_word, current_count))
        # reset counter and update current word
        current_count = count
        word = word
# output for last word
if current_word==word:
    print('%s\t%s' % (current_word, current_count))
```

The streaming mode can be used with any programming language, as long as inputs are read from the standard input and outputs are written to the standard output stream. Technically, the streaming mode is just a Hadoop MapReduce

implementation that is bundled as a jar-Archive and can be used. For example, the following command could be used to run the word count.

```
hadoop jar hadoop-streaming.jar \
  - input myInputDirs \
  - output my OutputDir \
  - mapper mapper.py \
  - reducer reducer.py \
  - file mapper.py \
  - file reducer.py
```

The hadoop-streaming.jar is a normal MapReduce implementation of MapReduce in Java, as we have it explained above. The map() function of this jar calls the mapper.py within its map() function and the reducer.py within its reduce function() and communicates with the python scripts via the standard input/output. The input and output parameters define where the data is read from and where the results will be written to in the HDFS. The final two parameters are required, to make the Python scripts available on the compute nodes.

12.3.6 Additional Components

In addition to the core components we have shown above, there are several other components, that we do not discuss in detail, but want to mention regardless. We briefly want to discuss two of these additional components.

Combiner functions are similar to reduce() function, but are running locally on each DataNode and are executed before shuffling the data to another node. In many use cases, the combiners are identical to reduce(). In the word count example, we can apply reduce() multiple times in a row without problems, since the reducer function creates the sum of the values of the keys. Without a combiner, these values will always be exactly one. With a combiner, we would create a word count locally on each DataNode, then shuffle the word counts of the nodes to reduce() function, that would create the total word count from the word counts on each DataNode. Combiners can, therefore, reduce the traffic between nodes by aggregating results prior to the reduce() function. For example, we would shuffle one pair instead of two pairs to the reducer, thereby cutting the traffic in half. The larger the amount of data that is handled locally at each node in the Hadoop cluster, the larger the potential for reducing the traffic with a combiner. The requirement for chaining functions this way is that the functions are *idempotent*.

The *MapReduce Job History Server* is another user-facing component of a Hadoop deployment. Users of a Hadoop cluster can access the Job History Server to access information about submitted Hadoop application. This information includes log files of the job execution, start and end times, as well as the state of the job, e.g., whether it it pending, running, finished, or failing.

12.3.7 Limitations

Hadoop has two major limitations. The first is that if an algorithm requires multiple map() and/or reduce() functions, their order of execution must be manually specified through the creation of `Job` objects and through the definition of their dependencies. This means that we cannot just add multiple map() or reduce() jobs in a specific order to a Job object, but instead have to define multiple Job objects, specify which Job object needs to wait for which other objects, to which Job object the output of a Job object should possibly be forwarded as input, etc. This manual modeling of dependencies is error prone and mistakes are hard to spot.

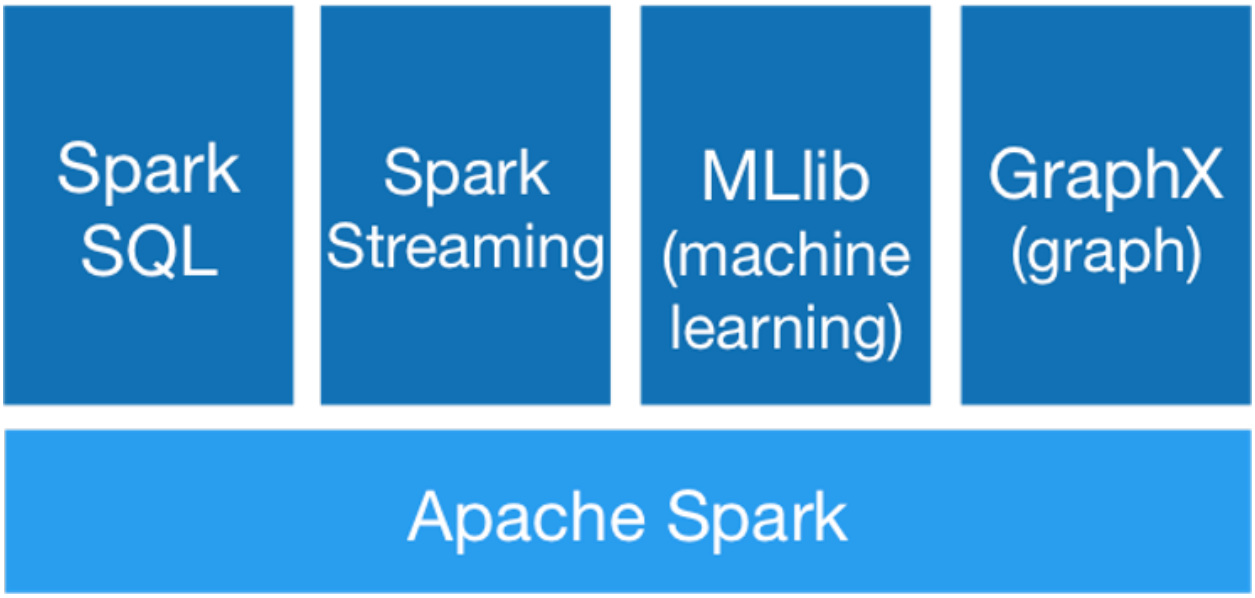
Related to this limitation is that the jobs always have to write their intermediary results to the HDFS. While this is not a problem is the data only needs to be read and processed once, this is severe overhead if multiple computations on the same data are performed, because the results are not kept in-memory. Thus, iterative algorithms are not very efficient with Hadoop.

12.4 Apache Spark

Apache Spark is another framework for working with Big Data that was created with the motivation to overcome the limitations of Hadoop: in memory data analysis, such that intermediary results do not have to be written to the HDFS and the convenient support for complex algorithms with many map() and reduce() functions.

12.4.1 Components

The first difference of Spark in comparison to Hadoop is that Spark contains a powerful software stack for data analysis, while Hadoop only provides basic functionality, which is often used by other software to provide these functions. Apache Spark consists of core components that are responsible for the Big Data processing. On top of these functionality, Spark provides libraries that simplify the definition of data analysis tasks.



Spark SQL suppers SQL-like queries to the data that is used within analysis. Because SQL is such a widespread language for working with data, this allows users to load and manipulate data with Apache Spark without a steep learning curve. *Spark Streaming* enables the live processing of *streaming data*. This way, Spark can handle continuously incoming data, which is another limitation of Hadoop. With the MLlib and GraphX, Spark also contains two libraries for data analysis. MLlib provides implementations for many of the algorithms we saw in the last chapters. GraphX is for processing of graph data, e.g., social networks.

12.4.2 Data Structures

The data structures of Spark are designed for in-memory data processing, which is in contrast to the HDFS approach by Hadoop where file operations are used. At the core, data is contained in *Resilient Distributed Datasets* (RDDs). These data structures provide an abstraction layer for data operations, independent of the actual type of the data. The RDDs organize data in immutable partitions, such that all elements in a RDD can be processed in parallel. Thus, the RDDs are similar to the key-value pairs of Hadoops MapReduce. Consequently, Spark allows the definition of map() and reduce() functions on RDDs, given that the data within the RDD are key-value pairs. However, Spark goes beyond the capabilities of Hadoop, because RDDs can also contain data that are not key-value pairs, filtering of RDDs is supported and, in general, any user-defined function can be applied to the data within an RDD. Additionally, the RDDs can be persisted to the storage, if requested.

Since Spark 2.0, users of Apache Spark do not have to work with RDDs anymore, but can also use data frames, similar to the data frames we know from pandas. This data frame API is rightly coupled with the Spark SQL components, which is the primary way for creating the data frames. The support for data frames is another reason why getting started with Spark is often simpler than with other Big Data frameworks.

12.4.3 Infrastructure

While Spark contains all components to create a compute cluster, this is not the normal way in which Spark is used. Instead, Spark is usually used together with one of the many compatible technologies. For example, Spark is fully compatible with YARN and HDFS, i.e., data stored in Hadoop clusters can also be analyzed with Apache Spark. Spark can also be used with many other technologies, e.g., EC2 clouds or Kubernetes for computation or databases like Cassandra, HBase, or MongoDB for storage. Thus, analysis designed with Apache Spark are not locked into a specific framework, which is another notable difference to Hadoop.

12.4.4 Word Count with Spark

While Spark itself is written in Scala, Spark applications can also be written in Java, Python (PySpark), and R (SparkR). Users do not have to define the dependencies between different tasks manually. Instead, Spark assumes that the order in which the tasks are defined is the order in which they should be executed. If tasks are using different data, they can be executed in parallel, if a task requires data from a previously defined task, this data is automatically shuffled to the task and the execution only starts when the data is available. Due to this, the source code for Spark jobs contains less boilerplate code for defining formats than that of Hadoop applications. For example, with Apache Spark, the word count my look like this.

```
# sc is the SparkContext, which is similar to the Configuration of Hadoop
text_file = sc.textFile("hdfs://data.txt")
# flatMap can input map the input to multiple outputs
# map maps each input to exactly one output
# reduce by key is the same as reduce in Hadoop
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://wc.txt")
```

Thus, complex tasks can be implemented easier with a framework like Spark.

Note:

Python `lambda` functions are anonymous functions that can expressed in one line. For example, `lambda a, b: a+b` would be the same as defining and calling a function

```
def fun(a, b):
return a+b
```

12.5 Beyond Hadoop and Spark

With Hadoop and Spark we present to popular technologies for Big Data analysis. However, due to importance of Big Data in the last decade, there are also many other important technologies, e.g., databases that are optimized for Big Data, alternatives for Stream processing of data, or tools for the management of compute clusters. There are whole ecosystems around Hadoop, Spark, and other technologies, that are often to some degree compatible with each other. This tool landscape is still evolving and constantly changing, even though some core technologies, like Hadoop or Spark for computations, or databases like Cassandra have now been important parts of the market for many years. Regardless, a complete discussion of tools is out of our scope.

A strong aspect of Big Data is that many of the state-of-the-art technologies are open source, meaning that often there are no licensing costs associated with using the software. Regardless, using these tools is still often not cheap, because the resources for running Big Data applications are expensive.

APPENDIX

A

APPENDIX

Mathematical Notations

Notation	Definition
$\mathbb{R}$	Real space, i.e., more or less any numerical value.
$\mathbb{N}$	Natural numbers, i.e., any integer greater than 0.
O	Object space, i.e., a set of real-world objects.
ϕ	Feature map, i.e., a map that defines the values of the features for objects.
$\mathcal{F}$	Feature space, i.e., the values of all features. Often the $\mathbb{R}^d$, i.e., the d-dimensional real space. In this case there are $d \in \mathbb{N}$ features.
X (clustering, classification, regression)	Used for the instances of objects in the feature space. Depending on the context, X is either a set of instances have $X = \{x_1, ..., x_n\} \subseteq \mathcal{F}$. There are also some cases where X is used as a random variable instead, the set would then be n realizations of this random variable.
Y (clustering, classification, regression)	Used for the value of interest, e.g., the classes for classification or the dependent variable in regression. Defined either as a set $Y = \{y_1, ..., y_n\}$ or a random variable (see X).
I	Finite set of items {i_1, . . . , i_m}.
T	Finite set of transactions $T = \{t_1, ..., t_n\}$ where $t_i \subseteq I$ for $i = 1, ..., n$.
X (association rules)	Antecedent of an association rule.
Y (association rules)	Consequent of an association rule.
$X \Rightarrow Y$	Association rule where Y is a consequent of X.
$\binom{n}{k}$	The binomial coefficient $\binom{n}{k} = \frac{n!}{(n-k)!k!}$.
$\mathcal{P}(I)$	The power set of a finite set I.
$\lvert \cdot \rvert$	Cardinality of a set, e.g., $\lvert X \rvert$ for the number of elements of X
$d(x, y)$	Distance between two vectors x and y, e.g., Euclidean distance, Manhattan distance, or Chebyshev distance.
$argmin_{i=1,...,k} f(i)$	The value of i for which the function f is minimized.
$argmin_{i \in \{1,...,k\}} f(i)$	Same as $argmin_{i=1,...,k} f(i)$.
$\min_{i=1,...,k} f(i)$	The minimal value of the function f for any value of i.
$\min_{i \in \{1,...,k\}} f(i)$	Same as $\min_{i=1,...,k}$.
$argmax$	See $argmin$.
$\max$	See $\min$.
$\sim$	Used to define the distribution of a random variable, e.g., $X \sim (\mu, \sigma)$ to specify that X is normally distributed with mean value μ and standard deviation σ.
C (classification)	Set of classes.
C (clustering)	Description of a cluster.
h	Hypothesis, concept, classifier, classification model.

continues on next page

Table 1 – continued from previous page

Notation	Definition
h^*	Target concept.
h'_c	Score based hypothesis that computes the scores for the class c
$P(X = x)$	Probability that the random variable X is realized by the value x.
$p(x)$	$p(x) = P(X = x)$ for a random variable x.
$P(X\|Y)$	Conditional probability of the random variable X given the random variable Y.
$H(X)$	Entropy of the random variable X.
$H(X\|Y)$	Conditional entropy of the random variable X given the random variable Y.
$I(X;Y)$	Information gain for X/Y if the other variable is known. Also known as mutual information.
e_x	Residual of a regression.
x_t	Values of a time series $\{x_1, ..., x_T\} = \{x_t\}_{t=1}^T$.
T_t	Trend term of a time series.
S_t	Seasonal term of a time series.
R_t	Autoregressive term of a time series.