COURSE BOOK

Case Study: Model Engineering

DLMDSME01

INTERNATIONAL UNIVERSITY OF APPLIED SCIENCES

Course Book

Case Study: Model Engineering

DLMDSME01

INTERNATIONAL
UNIVERSITY OF
APPLIED SCIENCES

Masthead

Publisher:
IU Internationale Hochschule GmbH
IU International University of Applied Sciences
Juri-Gagarin-Ring 152
D-99084 Erfurt

Mailing address:
Albert-Proeller-Straße 15-19
D-86675 Buchdorf

media@iu.org
www.iu.de

DLMDSME01
Version No.: 001-2022-0817

Module Director

Dr. Markus Pak

Mr. Pak has been teaching in the Master Data Science program at IU International University of Applied Sciences since 2020 with a focus on model engineering and machine learning.

Mr. Pak studied physics at the University of Graz, specializing in computer-oriented and theoretical physics. He received his PhD from the Eberhard-Karls-University of Tübingen with a thesis on the mechanism of mass generation after the Big Bang. After a research position on a project financed by the Austrian Science Fund (FWF), Mr. Pak moved to the private sector, where he worked on the application of Monte Carlo simulations and stochastic modeling to various business fields.

Table of Contents

Case Study: Model Engineering

Module Director 3

Introduction
Case Study: Model Engineering 7

Signposts Throughout the Course Book 8

Learning Objectives 9

Unit 1
Data Science Methodology 12

1.1 CRoss-Industry Standard Process for Data Mining (CRISP-DM) 12

1.2 MS Team Data Science Process (TDSP) 22

Unit 2
Data Quality 26

2.1 Evaluating Data Quality 26

2.2 Using Low Quality Data 32

2.3 Data Duality and Domain Knowledge 34

Unit 3
Feature Engineering 40

3.1 Building New Features 40

3.2 Splitting Variables 50

3.3 Feature Engineering Exploiting Domain Knowledge 54

Unit 4
Feature Selection 58

4.1 Univariate Feature Selection 60

4.2 Model-Based Feature Selection 63

Unit 5
Building a Predictive Model 68

5.1 Establishing a Benchmark Model 74

5.2 Prediction as Probabilities 76

5.3 Interpretable Machine Learning and Results 78

Unit 6
Avoiding Common Fallacies 84

6.1 Overtraining and Generalization 84

6.2 Overfitting and Occam's Razor 97

6.3 Workflow Automatization and Model Persistence 99

Appendix 1
List of References 108

Appendix 2
List of Tables and Figures 120

Introduction

Case Study: Model Engineering

INTERNATIONAL
UNIVERSITY OF
APPLIED SCIENCES

Signposts Throughout the Course Book

Welcome

This course book contains the core content for this course. Additional learning materials can be found on the learning platform, but this course book should form the basis for your learning.

The content of this course book is divided into units, which are divided further into sections. Each section contains only one new key concept to allow you to quickly and efficiently add new learning material to your existing knowledge.

At the end of each section of the digital course book, you will find self-check questions. These questions are designed to help you check whether you have understood the concepts in each section.

For all modules with a final exam, you must complete the knowledge tests on the learning platform. You will pass the knowledge test for each unit when you answer at least 80% of the questions correctly.

When you have passed the knowledge tests for all the units, the course is considered finished and you will be able to register for the final assessment. Please ensure that you complete the evaluation prior to registering for the assessment.

Good luck!

Learning Objectives

The course "**Model Engineering**" focuses on how to build machine learning models for data science and artificial intelligence and is aimed as a practice guideline discussing many issues that arise in the day-to-day work of a data scientist.

First, we discuss two common methodologies that are commonly applied in the context of model building: the Cross-Industry Standard Process for Data Mining (CRISP-DM) that originates in the data mining community and the more recent Team Data Science Process (TDSP) proposed by Microsoft.

Before training any machine learning model, the quality of the data has to be assessed. This is critical as the machine learning algorithms learn from the data they are given and any issues in the data may result in degraded performance. We discuss specific techniques to handle situations where the quality of the data is low.

One of the most important aspects in building machine learning models is to devise and construct new feature variables that exploit the data optimally and include domain knowledge where possible. First, we discuss a range of techniques to create new features and then focus on how to select the most useful ones. Specific focus is given to controlling the false discovery rate.

We also discuss a multitude of aspects relating to building machine learning models in practice such as how and why to build hierarchical or ensemble models, why we should establish a benchmark model as a first step, and how to create interpretable machine learning models instead of "black box" approaches.

Finally, we discuss a range of common fallacies and how to avoid them such as overtraining, overfitting, or generalization of the model, as well as how to approach operational issues of bringing a model into a production environment.

Unit 1

Data Science Methodology

STUDY GOALS

On completion of this unit, you will have learned ...

... which different standard processes for data analysis exist.

... why we should follow such a process in our analysis work.

... how to identify the relevant parts of the processes and apply them to original analyses.

1. Data Science Methodology

Introduction

A key aspect of starting a new data science project is to organize the project outline and workflow. Once we start, we can assume that the definition of the use case has already been done. We know what we want to do and why, how to evaluate success, and what the outcome should be, e.g., a production level system, gaining knowledge, or the output of a predictive model which is then used in another project. We then need to organize how we work in the project in terms of project management, such as waterfall (Benington, 1956, 1983), **Scrum** (Pries & Quigley, 2011), or **Kanban** (Ohno, 1988). In this unit, we will focus on the methodology to follow, addressing the questions about which steps need to be taken in the project, in which order, and how they are related to each other. One of the first methodologies developed was Knowledge Discovery in Databases (KDD) (Piatetsky-Shapiro, 1990), which focused on the use of databases as almost all usable information was stored in these systems. The CRoss-Industry Standard Process for Data Mining (CRISP-DM) was later developed as an open standard for knowledge extraction data mining (Shearer, 2000). More recently, Microsoft has proposed the Team Data Science Process (TDSP) (Microsoft, 2017) that includes more of the requirements, challenges, and procedures found in current data science projects.

Scrum and Kanban
These are two agile development methods. In contrast to conventional project planning, development goals of the project are iteratively adjusted in Sprints, each lasting about two weeks. The alignment of the development team with the priorities of the stakeholders is monitored by a product owner.

In this unit, we explore in more detail how to use the CRISP-DM and TDSP to organize a new project and which steps we need to consider.

1.1 CRoss-Industry Standard Process for Data Mining (CRISP-DM)

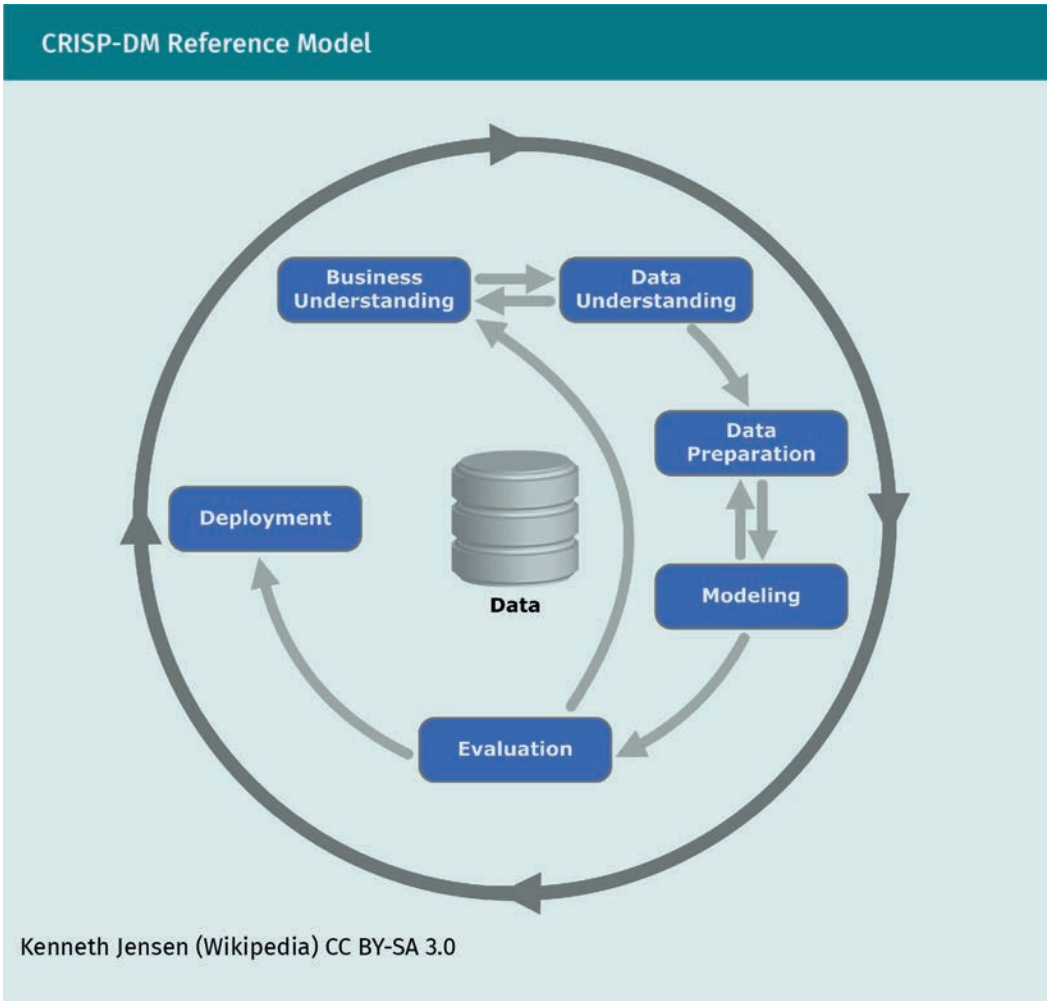
This standard was developed in the 1990s, when the amount and type of data had started to grow significantly, later refered to as “big data” (Laney, 2001). This increase in amount and variety of data also led to many technical challenges in handling and processing the data. However, apart from individual practices established in single companies, there was no common methodology or approach for how to extract insights or knowledge from data and tie the use of data to business goals. The project “CRISP-DM” was funded by the European Union between July 1997 and December 1998 with the objective to “cater for data mining needs of industrial users of huge data warehouses, by providing an industry-neutral and tool-neutral process model. This project will develop a data mining process which is fast, well understood, reliable, and valid across a wide range of applications” (NCR Danmark, 1998, para. 1). The project team was comprised of members from the companies NCR Danmark A/S (Denmark), Daimler-Benz AG (Germany), Integral Solutions Ltd (UK), and OHRA (Netherlands). Each company contributed a specific skill set, augmenting the others: NCR is a large computer company known for the Teradata data warehouse; Daimler-Benz was already establishing data mining approaches; and Integral Solutions had developed the SPSS statistical modeling package that was later acquired by IBM (Sloman, 2009).

The resulting CRISP-DM process described in Shearer (2000) is now, to some extent, abandoned. The Web page associated with the project is still owned by IBM but no longer provides any content or reference material, although IBM itself has included some of the methodology in the commercial IBM (2012) SPSS Modeler. Nevertheless, the process gives valuable insights into which steps are required in any data analytics or data science project. Moreover, the CRISP-DM approach is intentionally relatively generic, and each step should be interpreted as a "headline" under which a wide range of topics specific to the problem can be summarized.

The CRISP-DM process is summarized in the graphic below and consists of the following six steps:

1. Business understanding
2. Data understanding
3. Data preparation
4. Modeling
5. Evaluation
6. Deployment

Arrows indicate which of the processes are linked either with each other or where dependencies exist. Arrows that point in both directions indicate that the relevant items can influence each other. For example, a deeper understanding of the data can lead to a revised business understanding and vice versa. The arrows encircling the process indicate that in most practical applications the process has to be considered iteratively and continuously as the project advances.



Business Understanding

The CRISP-DM process was developed from an industry perspective. Unsurprisingly, the starting point of any project in this methodology focuses on a detailed understanding of the business context. The main objective of this point is to understand how the project fits into the overall business perspective and what should be gained from this project. This phase should end with the creation of a first project plan detailing the resources that are likely needed, any specific requirements, assumptions, constraints, risks, or contingencies, as well as tools and techniques used in the project (Shearer, 2000). This step is closely related to the identification of **use cases**.

Data Understanding

Having formulated a first project plan and an understanding of the business objectives of the project, the second stage focuses on an initial understanding of the data. Before this exploratory phase can begin, all relevant data—insofar it is known which data are needed following step one—are acquired and made accessible. If data have to be

accessed from a wide range of sources with different access protocols, it is beneficial to centralize the data into a common data pool as much as possible or permissible. Furthermore, any regulatory or access criteria need to be followed. In case of sensitive or personal data, further security measures need to be put into place to be able to work with the data. In some cases, it may be necessary to anonymize the data before further processing can start.

Once all data are available, we can start to **explore the data**. In the beginning, an initial data quality assessment is helpful. These steps might include an initial evaluation if all data fields are filled, if and how many values are missing, if the type of the data obtained from the sources conform to the specification and description of each data source, or if there are any systematic deviations that may point to faults in the data acquisition, processing, or transfer mechanisms. A more detailed understanding of the quality of the data will come once the data are explored more fully; however, this preliminary assessment will likely identify many significant obstacles that often need extensive attention to overcome.

In the next step, the data should be explored in more detail. To begin, we might create histograms of each variable, note their distribution, spread, or any outlier. Next, combinations of any two variables can be considered, the correlation between variables or the behavior of variables in terms of other variables. In many cases, a more detailed understanding of the data might lead to a revision of the business objectives. For example, the data might not support a particular business idea or the data available are not sufficient to fulfill the overall aim of the project. It may be necessary to update the business understanding or scope of the project.

Use case
The use case is the definition of a (data science) project detailing how the project will create value, how this is measured, which data are going to be used, and how a prediction is transformed into an actionable decision, creating the value of the project.

Explorative Data Analysis (EDA)
When little about the data is known, EDA is used to learn details about the data, relationships, and correlations between variables to gain a deeper understanding of the data used in a project. Consequently, this allows more questions to be "asked" of the data, leading to further insight.

Data Preparation

This step includes all activities that are required for the next step in which the predictive model is being built. This step includes the selection and formatting of the data for further modeling and processing, as well as a more detailed data quality assessment and data cleaning to make sure that all data are as well prepared as they can be (Shearer, 2000).

Modeling

In this phase, the model used for the project is developed. The process starts with selecting the modeling technique to be used, tuning any parameters the model might have to describe the available data correctly. The model should be tuned or trained on a fraction of the data, also called the training data and then be verified on an independent dataset called the test data that are not used in the training process. In practice, these two datasets can be obtained by randomly assigning about 80 percent of the data to the training data while retaining 20 percent for later testing.

Evaluation

The evaluation step is crucial to understand whether the business objectives outlined in the first step have been reached by the project. During this phase, the whole project is reviewed, and its findings compared to the original objectives and project outlines. A decision is now made whether the project has been successful, have its objectives been reached? In some cases, the evaluation step needs to refer back to the first step, the business understanding. If the original objectives have not been reached, the whole project may be revised, and the process restarts.

Deployment

The final step in the CRISP-DM process is the deployment of the project, for example in a production system where it runs on demand or continuously to offer a new product or service. This step includes planning the deployment both from a technical perspective, e.g., setting up the required resources and computers, connecting the data "pipelines" that feed data from acquisition to the predictive or modeling instance and feed the result into the required output or decision-making process. Furthermore, a detailed monitoring and alerting system should be set up to be able to intervene quickly in case of a disruption or degradation of model performance. Finally, the project should be well documented (Shearer, 2000).

Example
To illustrate the CRISP-DM process, we will use the California Housing dataset (Pace & Barry, 1997) which is often used in training exercises. The data can be obtained from Pace and Barry (1999). In this example, the data are set. In a real project, we would be able to add other sources of data as our understanding of the business objectives or the data improve.

The data consist of 20640 records and the following variables:

- the average house value (the target variable we wish to predict)
- average income
- housing average age
- average number of rooms
- average number of bedrooms
- population
- average population
- latitude
- longitude

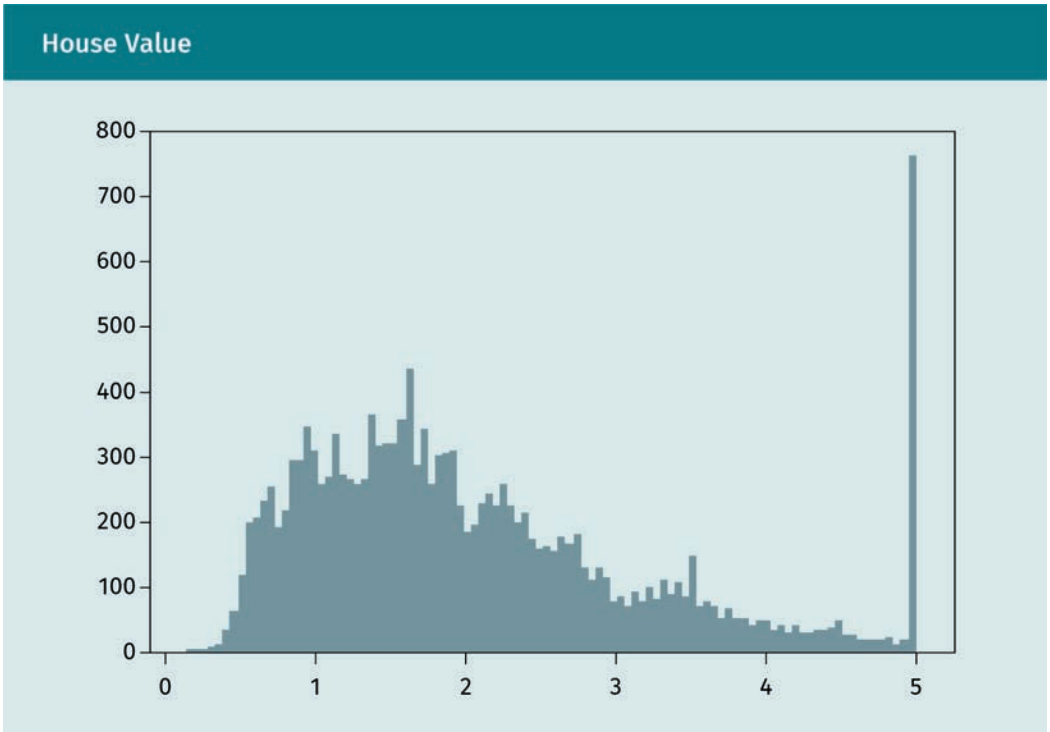
We start with the business understanding: In a real project we would focus on potential business objectives and how to achieve these using a data-driven approach. As the data are given in our example, we flip this question and investigate possible business

objectives we could formulate around these data. Without looking at the data, we can formulate a wide range of ideas based on the prediction of the house values, for example:

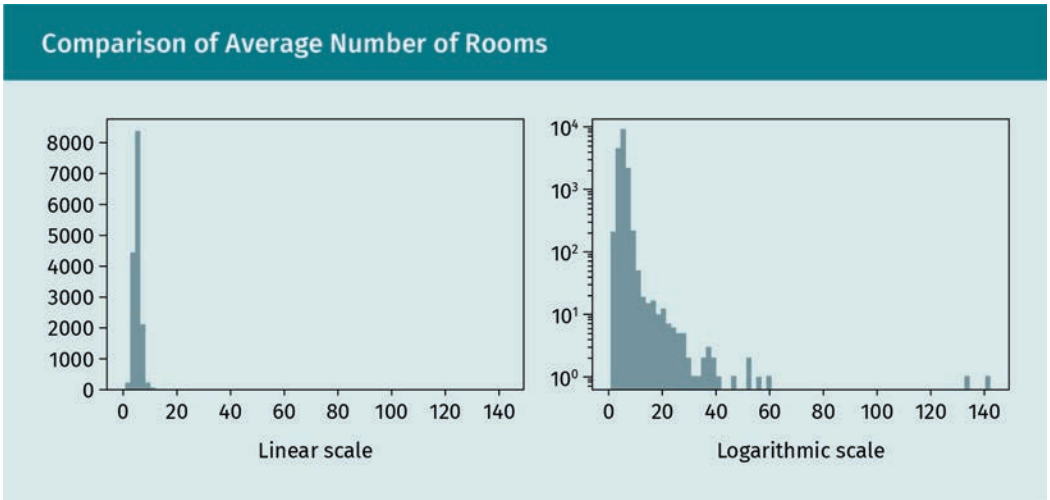
- As a seller, we might want to know the best price to sell our house.
- As a buyer, we might want to know the most realistic house price.
- As a real estate manager, we might want to quickly appraise a new property or advise new clients (either a seller or a buyer) on the value of a given property.
- As an app developer, we might want to provide users with a convenient and quick service if they want to buy or sell a house.
- As a bank, we might want to have a quick and convenient tool to assess lending requests (e.g., mortgage).

We can formulate a wide range of business objectives. We could refine each of these ideas further for a profitable niche for our business idea. For example, we could focus on affordable housing or premium houses, residential properties or commercial properties, and hotels or office spaces.

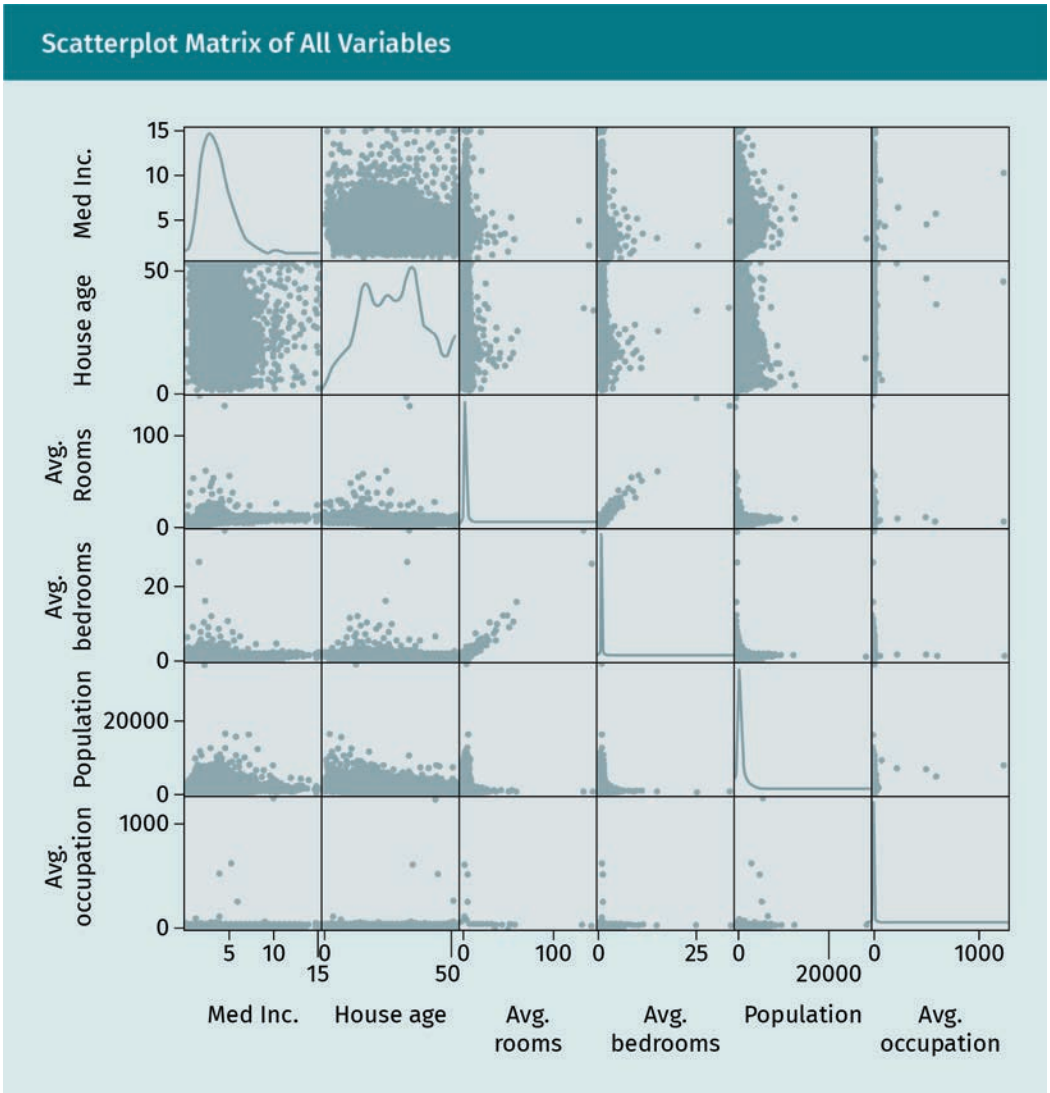
Next, we start exploring the data. After some initial data quality checks, we start by looking at the distribution of the target variable: the house value (shown in the figure below). The first inspection of this histogram reveals two surprises. First, the range is between zero and five, and then there is a strong peak at five. As we deal with house prices, we would expect the range of the variable to be in the tens to hundreds of thousands of dollars. Closer inspection of the data reveals that instead of the price, the logarithm was stored. This mathematical transformation doesn't change the properties of the distribution of data points; while this might be unexpected, it does not harm our project in any way. The strong peak (or spike) at the end of the distribution is more worrying. Indeed, a cutoff was introduced, and all prices exceeding a certain threshold were set to this threshold. Depending on our business objectives, this may or may not have an implication for the project. We need to revise step one. Given these data we are no longer able to focus on premium houses (e.g., large villas) or commercial estates (e.g., hotels or office buildings), as these likely exceed the threshold.



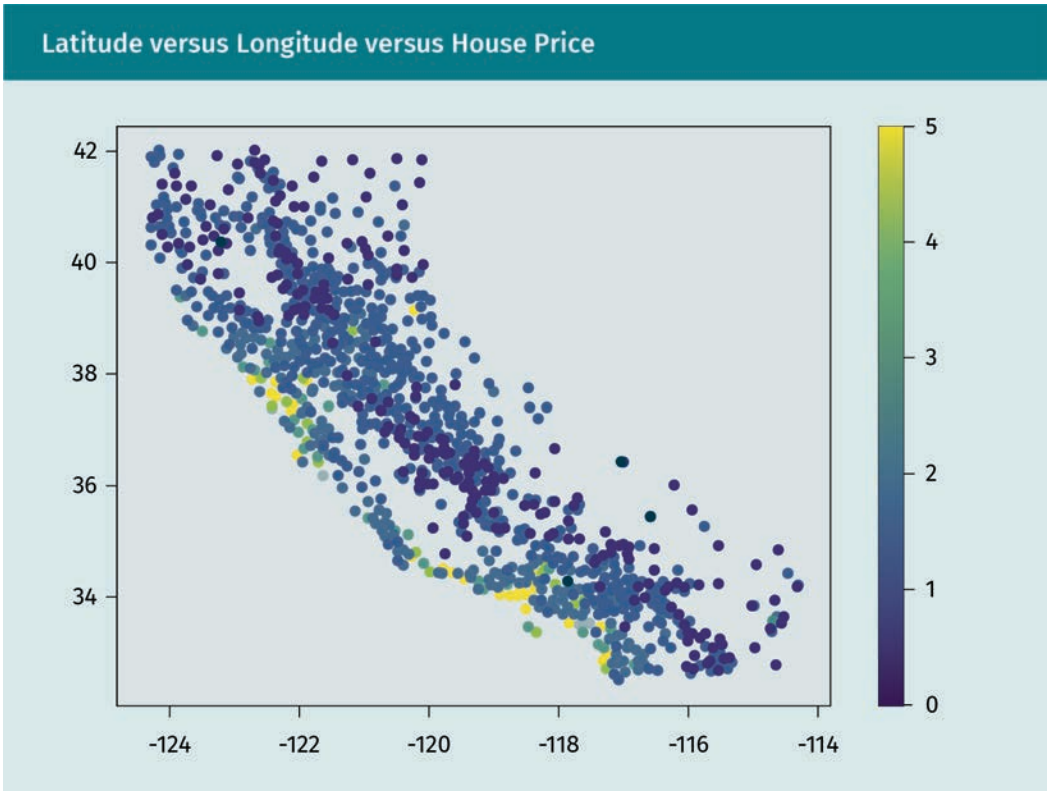
Another variable needing a more detailed investigation is the average number of rooms of houses in a given area, as shown in the comparison below. Looking at the distribution of this variable on a linear scale shows a narrow distribution around 5—10 rooms on average for houses in a given area. However, switching to a logarithmic scale on the y-axis, the histogram reveals a long tail towards higher numbers with a distinct peak just below 40 rooms and isolated data points up to 140 rooms. Without knowing more about the data, it is impossible to determine if these entries are related to data quality issues and simply wrong or belong to specific types of buildings, such as hotels or offices. In a real project, we would have to investigate further whether we could (or should) remove these data points from our dataset or if we need to revisit step one and refine our business objectives.



We can now investigate the correlations of all variables against each other in pairs (with the exception of latitude and longitude) as shown in the scatterplot. The distribution of each variable is shown on the diagonal of this matrix and a scatterplot of each variable against each other variable is shown in the off-diagonal fields of the matrix. As there are quite a few entries in each plot, gaining insights from such a visualization is a worthwhile, albeit difficult, endeavor.

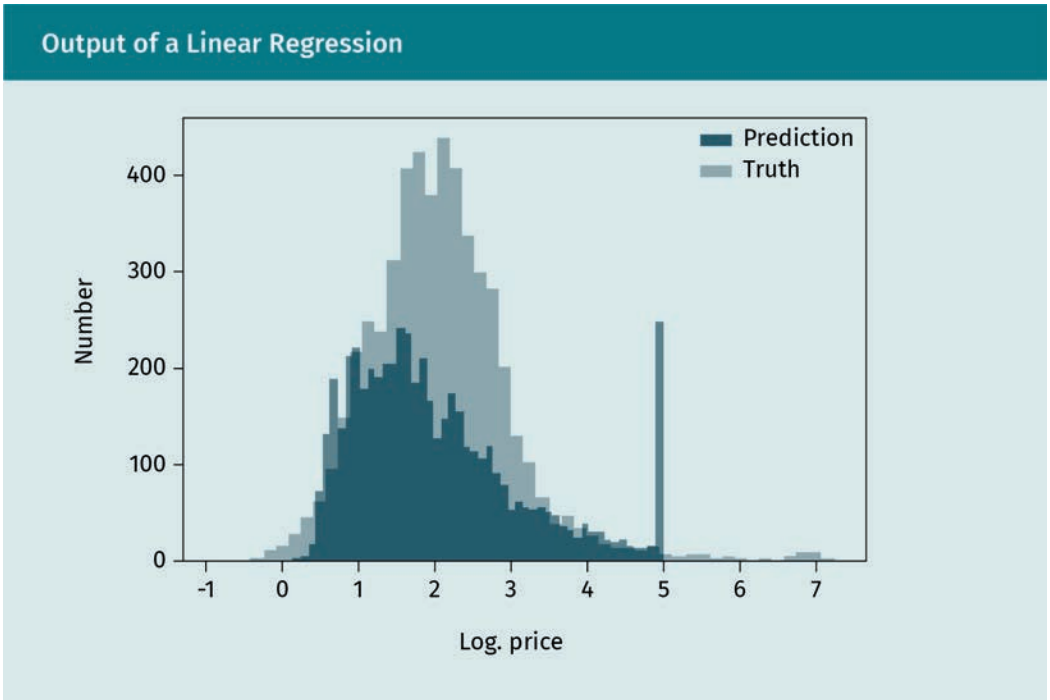


Finally, we investigate the variables "latitude" and "longitude." From experience, we know that these relate to coordinates. We show them in a two-dimensional graph and include the variable house price as a color index. As the graph below shows, the result is a map of California where the highest prices are found in cities such as San Francisco or Los Angeles as well as the Bay Area. Few to no settlements are found in desert areas —in line with our expectations. This type of visual provides powerful cross-check that the data make sense compared with our expectation or knowledge about the background of the project.



The next step in the CRISP-DM process is to prepare and clean all data, followed by the modeling step. As an example, we choose a linear regression to predict the price of a given house. However, as shown in the figure below, such a simple model is not successful; we would have to build a more sophisticated model. Once we are satisfied with the quality of our model, we need to make sure that it fits within the remit of our project outline and business objectives.

Finally, we deploy the model. Given our project outline, this could involve programming and publishing an app for a smartphone or integrating the prediction model into the mortgage advisory system of a bank.



1.2 MS Team Data Science Process (TDSP)

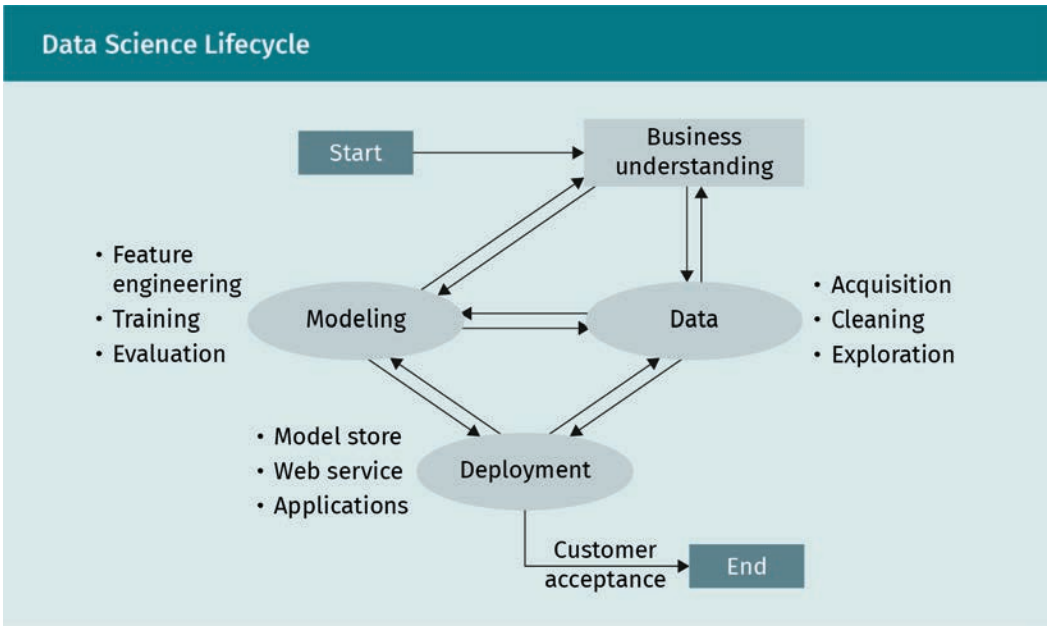
The Team Data Science Process (TDSP) was established by Microsoft (2017) as a synthesis of best practices established in data science-related projects at Microsoft and other companies. The TDSP consists of a description of the process itself, a description of relevant roles in a larger data science project team, and a collection of tools and utilities which might be used in the process.

Central to the Team Data Science Process is the Data Science Lifecycle. The Data Science Lifecycle has the following elements:

- Business understanding
- Modeling
- Data acquisition and understanding
- Deployment

Although not obvious at first glance, the lifecycle is closely related to the process outline of CRISP-DM from the previous section. The stage “Evaluation” in CRISP-DM is part of the modeling step in the Data Science Lifecycle; the stages “Data Understanding” and “Data Preparation” in CRISP-DM correspond to the item “Data Acquisition & Understanding” in the Data Science Lifecycle. However, there are also some notable differences between the two processes: Whereas the CRISP-DM process was envisioned as a process that can be started over, as indicated by the circle of arrows around the whole process, the Data Science Lifecycle has a definite beginning with a detailed Business Understanding as well as a definite end with the acceptance of the project by a cus-

tomer. Furthermore, all stages of the of the processes are connected with double-sided arrows, reflecting the incorporation of agile project management methods, such as Scrum (Pries & Quigley, 2011) that weren't widely available when CRISP-DM was developed.



In addition to the Data Science Lifecycle, the TDSP also defines a number of roles, tasks, templates (Microsoft, 2016b), utilities (Microsoft 2016a), and artifacts. Infrastructure components such as Git and Azure are strongly featured since the former is the quasi standard for version control while the latter belongs to the Microsoft cooperation.

The TDSP defines the following roles and tasks (Microsoft, 2017):

- Group manager, department head of all data science teams in a company
- Team lead, head of a dedicated data science team
- Project lead, head of a specific project
- Project individual contributor, executes the project. This position contributes to the concrete tasks. Contributors may have a wide range of backgrounds (e.g., Data Scientist, Business Analyst, Data Engineer) depending on the needs of the project.

The TDSP assumes that the project is developed in a Scrum setting (Microsoft, 2017), as such, there are four work items which occur frequently: Features, User Stories, Tasks, and Bugs. In the process, a Feature is a "project engagement," i.e., something which implements a specific aspect of a project for a particular client. The User Stories are then used to implement a specific feature. They are always considered to be end-to-end, i.e., contain all aspects such as data acquisition, data exploration and cleaning, model building, and deployment of the model within the remit of the user story and feature. The individual Tasks within a User Story correspond to the actual work that can

be assigned to an individual contributor. Bugs are urgent issues in either the documentation or the existing code and are treated separately to be able to escalate them above the other work items.

Strictly following the Team Data Science Process implies that a sufficiently large team is available to work in a Scrum setting effectively and that the team is prepared (and allowed) to base their work in the Microsoft Azure infrastructure. Nevertheless, if a data science project is to be executed with a smaller team than recommended for the Scrum process or with tools outside the Microsoft environment, the Team Data Science Process can still be used as an inspiration for how to organize a particular project.

Summary

Defining a methodology is a cornerstone of any data science project. Both the CRISP-DM and the TDSP define the core steps and their order. In particular, each project starts with a detailed business understanding, followed by a data step that includes the acquisition, understanding, exploration, and cleaning of the data. Subsequently, predictive models are created and evaluated before they are deployed in an application.

There are key differences between TDSP and CRISP-DM. TDSP has a definite beginning and end, whereas the CRISP-DM process is a bit more circular. The methods also differ in the specific definition of the main steps and their relation with each other.

Knowledge Check

Did you understand this unit?

You can check your understanding by completing the questions for this unit on the learning platform.

Good luck!

Unit 2

Data Quality

STUDY GOALS

On completion of this unit, you will have learned ...

... why curating and cleaning the data is of crucial importance in data-driven decision-making, machine learning, and artificial intelligence applications.

... how to identify and assess data quality issues.

... how to address the most common data quality issues.

... why using further or domain knowledge is important to curate and prepare data.

2. Data Quality

Introduction

Biased algorithm
The output of the algorithm is skewed towards a particular outcome even if this is a spurious relationship, e.g., if a data set was used to train the algorithm that is unsuitable for the task.

Data are the foundation on which all data-driven approaches, decisions, or models are built. This is particularly true for machine learning and artificial intelligence, as these use models that are trained on the data we data scientists select and prepare. Hence, if we are not careful, the algorithms and model may learn the wrong relationships or become severely biased. Watcher-Boettcher (2017) provides insight about the consequences of **biased algorithms** in real-world applications. It is therefore paramount that the data we use in any data-intensive application are curated carefully, any errors found in the data corrected, and—if possible—the underlying root causes identified and removed. Furthermore, we need to make sure that the data we use, e.g., in a machine learning model, represent the later use of this model well.

Data scientists reported that their daily work focused on data-related aspects such as collecting, organizing, and cleaning data takes up about 80 percent of their overall time spent on a data science project (Crowdflower, 2016). Accordingly, it is important to stress that preparing and cleaning data is a crucial aspect of building a data-driven or machine learning model. Only through deliberate action can we ensure predictions that are sensible and useful—or, more casually, "garbage in, garbage out."

2.1 Evaluating Data Quality

Before working with and exploring relationships in the data, the first step is to assess the overall quality of the data and to correct all errors. This is not a "heroic" act at the beginning of the project, but something that will persist throughout the effort. Each new data delivery, both for updates to the training data but also for the data that are going to be used in the later decision-making process, may contain new or different types of errors which haven't yet been encountered. Also, as the understanding of both the data and the project deepens, issues that were put aside in the beginning may become more important or profound as the project advances. Redman's (2004) case study on data quality emphasizes the strict data quality controls that have to be established to maintain a level of accuracy required by the project definition.

Assessing the quality of the data depends on the type of data we deal with. We can generally distinguish between unstructured and structured data.

Structured and Unstructured Data

Structured data are data which can be represented as a table in a fixed format, such as sales records of a shop, order books, student records, or marketing campaigns.

For example, structured data could look like this:

Structured Data (Sales Records)				
Date	Product ID	Store ID	Promotion Flag	Sales Amount
2019-01-02	123	456	false	5
2019-01-02	125	456	false	3
2019-01-04	345	789	true	10
2019-01-07	378	978	false	7

This example shows the (fictitious) sales of various products in several branches of a retail chain. Sales records are stored on a daily basis for each product and branch, along with the number of products sold, and whether these were the focus of a marketing effort during the sales period.

This simple example illustrates that data are often recorded with a specific granularity. Instead of recording each individual sale with a more detailed timestamp (e.g., including minutes and seconds), sales records in this example are aggregated to the level of days. This choice is typically made when the IT systems are designed and implies that any finer granularity is not accessible. For example, if a particular use case definition requires the availability of each individual sales record, this use case cannot be pursued unless the IT infrastructure is changed.

A particular type of data are time series-based data such as sensor readings during a manufacturing process. While each single measurement can be represented as a structured data point (e.g., timestamp, sensor ID, sensor value), the individual records are generally not independent from one another, as they refer to the same underlying process. For example, if a temperature sensor is attached to a kettle, the sensor would record the rising water temperature when the kettle is switched on, and the water is brought to a boil. Depending on how often the sensor is read, the expected temperature variation is quite small. If the kettle heats the water slowly, and the temperature is measured, for example, once per second, the difference between each measurement should be very small. These correlations between data points are generally higher if the frequency of the sensor readings is higher than the change of the physical process they monitor. While time series data can be stored in the same way as other structured data, it is worth considering whether dedicated data storage solutions are a better alternative.

In general, it would be preferable to keep data at the lowest granularity possible as it is always possible to aggregate data at a later stage; however, once data are aggregated, the finer underlying structure is lost. Storing data at a fine granularity also requires a large data storage and processing capacity to handle the large amount of data. The expense of this type of storage may prohibit it completely.

Unstructured data are data which cannot be represented by a fixed format or table. Examples for unstructured data are images, audio recordings, and text.

In many applications, unstructured data cannot be used directly but specific features, such as the topic or the mood of a text and the number of speakers in a dialogue, can be created automatically using deep learning approaches.

Metadata
This contains additional information about the provenance of the data, their content, or other details.

Unstructured data is often complemented with **metadata**. The metadata contains a range of details about the data or how it was recorded. For example, the recording of a telephone conversation is an example of unstructured data, whereas details such as the duration, telephone numbers of the participants, or time of the call are metadata which describe the details of how the telephone recording was obtained. However, this kind of metadata does not contain any information about the content of the recording itself.

Data Quality Evaluation

Assessing the quality of unstructured data strongly depends on the type of data and the context in which we want to use these data. A noisy or low resolution image, for example, may be acceptable in some scenarios where we focus on large and distinct aspects of the image but not in other cases where we are interested in finer details. We might evaluate anything from whether the colors of the image reproduce the original colors faithfully to whether the compression algorithm used in some image formats leads to any artifacts that are detrimental to the intended use. The same general considerations also hold for audio files. Generally, it is difficult to define a strict checklist for assessing the quality of unstructured data, as these types of data are very diverse. Instead, we should carefully determine at the beginning of the project which aspects of the data are important for the project and revise the list throughout the project.

With structured data, assessing the quality is—at least in the beginning of the process—a bit easier. Since all data are structured, they have to conform to a given format, i.e., all fields have a definite type and format. If the data are stored in a database, this is called the **database schema**. The first check is then to determine whether all fields are filled and whether the data conform to what is defined in the structure or schema.

For example, a data entry is missing in the following table:

Structured Data (Missing Information)				
Date	Product ID	Store ID	Promotion Flag	Sales Amount
2019-01-02	123	456	false	5
2019-01-02	125	456	false	
2019-01-04	345	789	true	10
2019-01-07	378	978	false	7

Database schema
Traditional databases store structured information in a given format that details which variables are present (columns) and which types they have. The schema of a database is the collection of all the details of how the various tables associated with a project are defined.

Even if this occurs only once in the data set, it is worth investigating whether or not there is an underlying fundamental reason. If this occurs only once, it might well be due to a random outage of some part of the data acquisition or processing system. If there are several instances of missing data, however, we should check for a pattern. Perhaps it always affects the same data source, occurs at a similar time, or other variables have similar value. Instead of the data or data source, the issue might be related to the load of the data processing machines if they are not able to process the data correctly or fully. In case of sensor data, missing values might also occur if the sensor is operated outside of its design specification. Perhaps the wrong type of sensor was used to handle a specific environment, thermal load, or other physical aspects related to the setup of the sensor or its environment. In each case, we should try to find the root cause of the missing values, fix it if possible, or notify the relevant departments. Once all missing values are found, it is helpful to assign them a special marker such as **NaN** to avoid leaving the relevant fields empty.

NaN
The acronym stands for "not a number." This special value is often used to indicate that a specific value of the variable is not filled with a "real" value or the actual value is not of a suitable type for this variable. For example, instead of the number 5, a data quality issue might be that someone has

The next fundamental check for structured data is to verify whether all data entries conform to the expected type of data. For example, a date field should always conform to a standard timestamp format, numerical fields should be integers or floating point numbers and not strings, etc. The last row in the following example shows an integer replacing a Boolean value.

Structured Data (Non-Conforming)				
Date	Product ID	Store ID	Promotion Flag	Sales Amount
2019-01-02	123	456	false	5
2019-01-02	125	456	false	3

entered the string "5"—which is not a numerical value.

Date	Product ID	Store ID	Promotion Flag	Sales Amount
2019-01-04	345	789	true	10
2019-01-07	378	978	false	7
2019-03-01	123	456	1	8

Boolean value
A Boolean value consists only of two stages such as 0/1 or true/false.

In the simplest case, this is due to using the integer "1" to represent the **Boolean** flag. Ideally, our defined structure of the data should have already either flagged or prevented this from happening. Perhaps the structure definition wasn't detailed enough, or it was exported to a format which doesn't prevent this. However, in this case something else might have happened. Rather than storing just a true/false flag whether a promotion was applied, at some point more information is stored in the same variable. For example, the "1" might not only signify that a promotion was applied, but the integer is used as a numeric code to specify which type of promotion was applied. Again, this should have been handled by changing the structure of the data accordingly; however, in many practical applications we find that a "quick fix" was applied to accommodate a recent change—and was never properly cleaned. Any such occurrence should be carefully noted and investigated.

The following table shows an example which is formally correct but might still point to a data quality issue.

Structured Data Example (Outlier)

Date	Product ID	Store ID	Promotion Flag	Sales Amount
2019-01-02	123	456	false	5
2019-01-02	125	456	false	–1
2019-01-04	345	789	true	10
2019-01-07	378	978	false	7

In the last column in the second row ("sales amount"), we notice that the sales amount is negative. From our understanding, a sales record should always be zero or positive—we can't sell less than a commodity. From our first understanding, and indeed the

description of the structure of the data, we are inclined to conclude that this field contains a data error, and we need to identify the root cause. However, the value might also be correct. In order to track returned products, these are assigned a “negative sales amount,” i.e., when a customer returns an item, it would be recorded as a “sale” of —1 units. Hence a sales amount of —1 might also mean that no product was sold, but one was returned. Without investigating further, we can’t determine whether the data are right or wrong. We need to find the root cause and change the structure of the data accordingly.

Further data quality issues which are often found in practical applications include the following:

- Interval-of-validity. Data are typically only valid for a limited amount of time; however, this valid range is often not explicitly specified. For example, a product is manufactured from a given list of raw materials. Even if the overall specifications of the final product don't change, the product itself is different if the raw materials are changed or the vendor providing the product is swapped. As this may lead to issues when the product is used, any changes made should be tracked.
- Tracking of interventions. Issues outside of the remit of the data are often not tracked. For example, a range of sensors record values in a manufacturing process. During maintenance, some of the sensors could be changed to a newer or different model which may behave differently in some ranges of the measured values. If these interventions are not tracked, the later model-building process will not be able to comprehend why some ranges of the data behave differently than others.
- Incomplete data history. Data storage systems are often designed so that the data are correct at any given moment but do not track each change that is made to the data. For example, a big company is restructured, and different parts of the company are now responsible for the manufacturing of products or different target markets, etc. Typically, great care is taken to ensure that all data storage systems and databases reflect the new responsibilities after the restructuring so that important aspects such as financial metrics are tracked accurately. However, if this migration does not consider the prior responsibilities, the historic data are nearly useless since no one will be able to see which metric was valid for which business unit at a given time in the past.

Any data quality issues found during the first investigation of the data should be noted in a detailed report and any root causes identified and fixed as much as possible. Furthermore, it is advisable to add an automated check for each data quality issue encountered. In many cases, new data deliveries will be made at a later stage of the project, or the intended use of the machine learning model requires regular new data deliveries. Any automated check we can implement now will help with operational use later.

2.2 Using Low Quality Data

White noise
This term is used to describe random frequencies of equal intensity.

Using low quality data in data-intensive applications, the development of machine learning models or artificial intelligence-based methods poses severe challenges and risks. In the best case, the data contain too much **white noise** which degrades the overall performance of what could have been achieved. In the worst case, the data contain or introduce a bias into further processing, or the data are not representative of the system they should describe. Biases can lead to many undesirable decisions, including racial bias or wrongful incarceration. Watcher-Boettcher (2017) provides other examples. It is important to remember that the underlying algorithm is generally okay, even if the chosen algorithm may not be the best choice or most efficient approach for a given project. The data with which the algorithm is trained taint subsequent predictions and decisions.

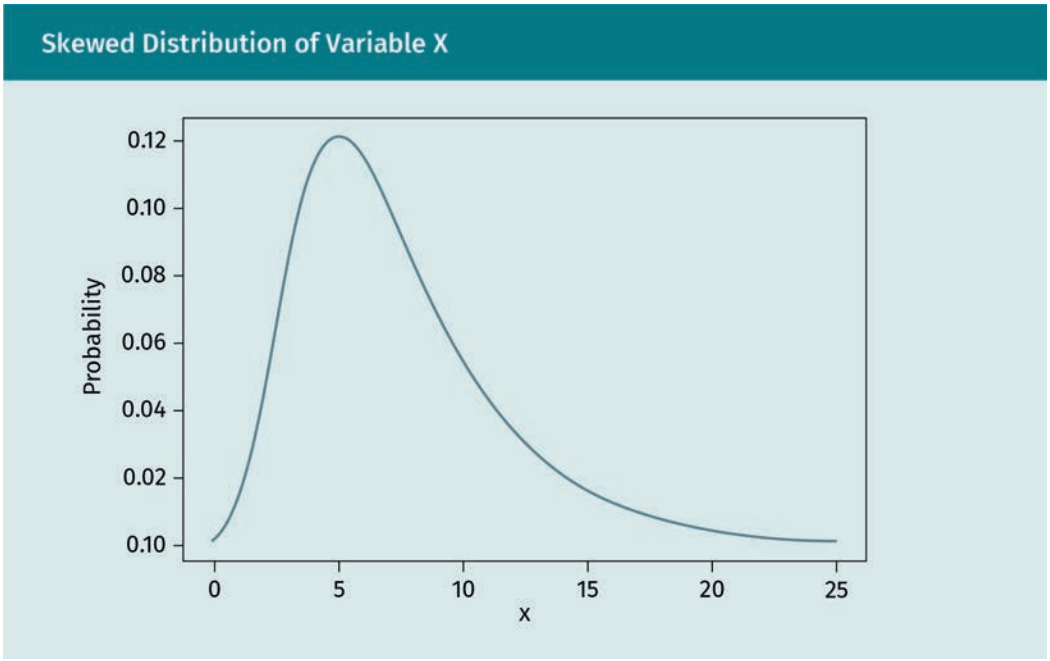
Any data quality issue that has been identified during the previously described assessment should be addressed and root causes eliminated. The content and the format of the values of each variable should be understood, and data structure (or the schema in case of the database) should reflect the (corrected) data accordingly.

Handling Missing Information

In many cases we will encounter missing values as in the table above. We need to find a way to address these as algorithms typically cannot handle missing values and expect that each variable is filled when processed in a predictive model. Before we define an approach to handle such occasions, it's important to remember that even this missing information can be informative. In an ideal world, we would have chosen a sensor where all possible working conditions are anticipated; however, in practical situations this may not be the case. The missing information indicates a special situation that we need to track. Hence, without knowing more, dropping all data records where one or more variable values are missing is generally not advisable as it may introduce a bias and information may be lost—in our case, the "special" situations where the sensor operates outside the specifications. Ideally, we would keep the missing information as "missing," indicated, e.g., by NaN or a special flag. However, very few algorithms can handle this; one example is NeuroBayes (Feindt & Kerzel, 2006).

Generally, we need to impute the missing values. The easiest approach is to replace the missing value with the mean of all observed values. In the case of skewed distributions, however, this is generally not ideal, as illustrated below. The mean does not reflect the shape of the distribution. Consequently, it is better to reconstruct the shape of the distribution of the specific variable. This can be done either analytically, if we know the underlying parametrization and can determine the parameters via a fit, or empirically by filling all values into a histogram. Then we can draw a random number from this analytical or empirical parametrization. However, replacing the missing value with the imputed one still removes the information that the value of the variable was originally

missing. This can be retained by adding a new variable which we set to 0 or *false* when no intervention has taken place and to 1 (or *true*) whenever we needed to change or impute a variable value.



Handling Imbalanced Classes

In classification problems, we are concerned with distinguishing whether a given event belongs to a specific class. In the simplest case, we need to separate events into two classes, e.g., A or B when a given image shows a cat or a dog. In multi-classification problems, we need to separate the events into multiple classes: A, B, C, etc. Typically, suitable training algorithms assume that the number of **training samples** available within each class represent the ratio of the classes observed in the system we want to describe. For example, if class A is twice as likely as class B, we would naturally assume that we have twice as many training examples of class A available compared to class B. However, this may lead to complications when one of the classes is much more likely than the others.

Say class A dominates the dataset used for training the algorithm, and there are only a few instances of the other classes B, C, etc. In this case, the algorithm would likely learn that returning "class A" is the right answer in any situation since the error attributed to the few cases of class B, C, etc., is very small. This type of algorithm would not have learned a specific property of class A but have returned this class, as its *a priori* probability is much higher than for the other ones.

Training data
These data are used to build the model, such as a machine learning algorithm. The test data are an independent data set that has the same properties as the training data but have not been used during the training of the algorithm and can be used to evaluate the performance of the algorithm.

For example, we might be tasked with identifying a faulty machine. Since all machines typically run well, the default case "not faulty" is much more likely than the few exceptions of the case "machine broken or about to break." However, since we want to identify these cases, returning "not broken" per default would render the algorithm useless. In other cases, we might be faced with the challenge that samples of, say, class B are much easier to obtain than others. In this case, our training dataset would contain a larger number of samples compared to the other classes, even if this doesn't reflect the relative composition of the classes in the physical system. To balance the classes, we can follow one of these approaches:

- Undersampling. If we have many data points in our overall data sample, we can choose the number of training samples randomly within each class such that the overall training dataset is either balanced or reflects the relative size of the classes we need.
- Oversampling. In the case of few data points, we may feed the data in the underrepresented classes to the algorithm more than once during the training process. However, as these are exactly the same data points, it may help the training of the algorithm—but only up to a point since no new information is added.
- Data augmentation. We can transform the data slightly so that they are presented to the algorithm during training in a different way, e.g., in case of images we can use different parts of the image or crop, rotate, or use other simple transformations.
- Synthetic data. In some cases it may be possible to create artificial data that looks identical to the "real" one, for example, using generative adversarial networks (GAN) (Goodfellow et al., 2014). Before using this approach, we would have to verify that the synthetic data is sufficiently similar to the "real" data to be used for training other algorithms to avoid introducing a bias.

2.3 Data Duality and Domain Knowledge

We often hear "Let the data speak for themselves" (Seamen's Church Institute, 1915; Anderson, 2008; Zhu, 2014; Gould, 1981). This school of thought posits that the data should be understandable on their own, without necessitating further or prior information.

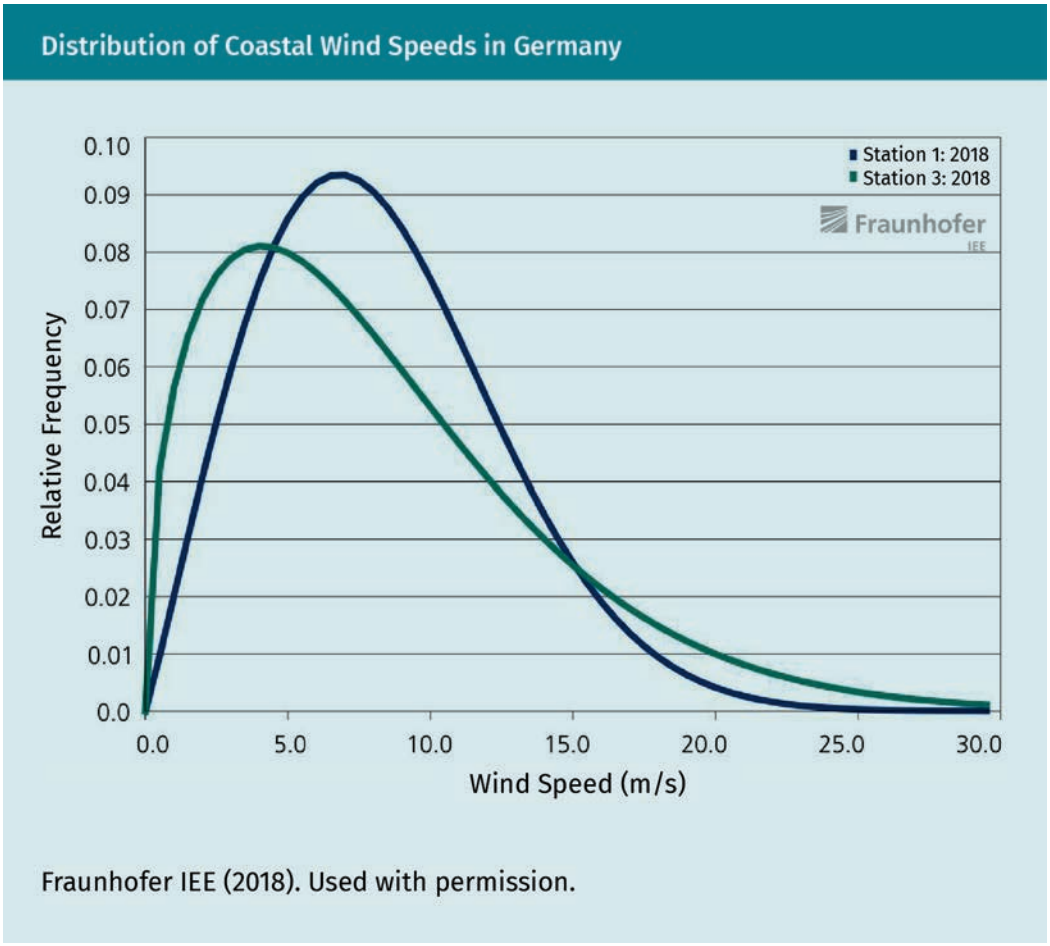
In reality, pure numbers mean little on their own. This is known as "data duality" or "information duality." Smith (2007) states that "in order for data to have meaning, it needs two complementary components" (Duality section, para. 1). We need the raw numbers and then some "understanding mechanism" by which the data can be interpreted and obtain some meaning.

In the context of data-intensive sciences, we stress that the data are acquired within a given (application) domain or use case. The data are not taken as abstract entities but come from concrete systems, applications, or use cases. Likewise, any predictions or subsequent decisions made by analytical, statistical, or machine learning/AI models using the data will also be applied within a specific remit or domain. Ergo, any domain

knowledge we have about the content of the data and the use case and how the data are recorded or processed and stored is useful, as well as the counsel of subject-matter experts.

When considering data quality, a good example of useful domain knowledge is the treatment of **outliers**. Generally speaking, any value that is far from the bulk of the data can come about because either the data acquisition and processing mechanisms resulted in an erroneous record, or the queried value is a valid measurement that occurs very infrequently. This is illustrated in the figure below which shows the distribution of wind speeds on the northern coasts of Germany with the Baltic and Northern Seas. The distributions are known to follow a Weibull distribution, and the observed range in 2018 was between zero and 30 m/s.

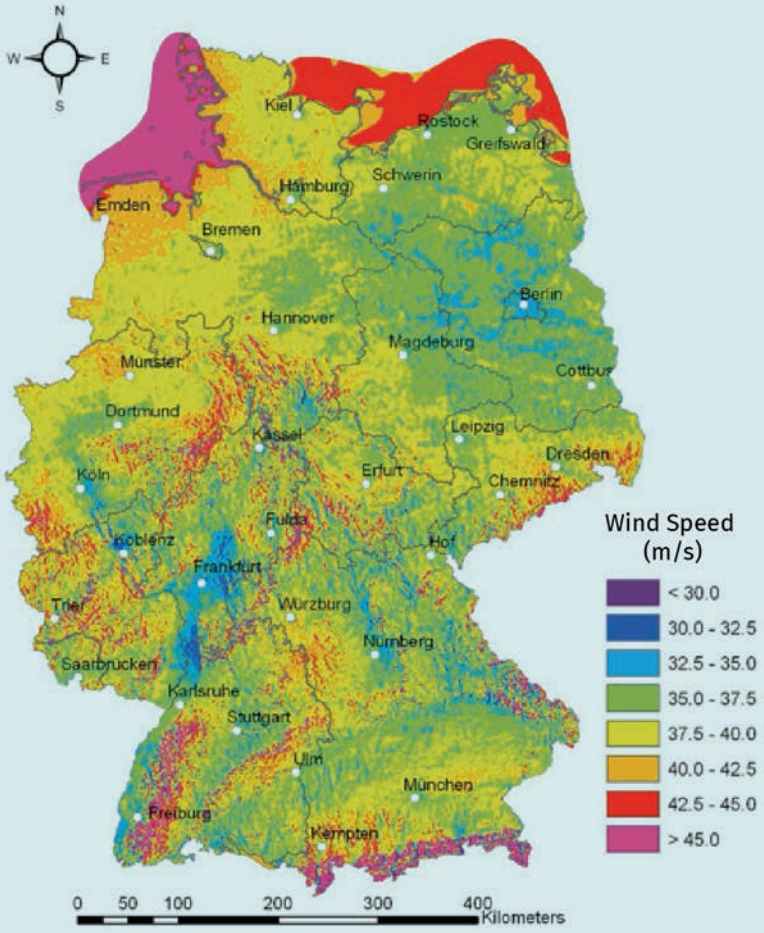
Outlier
This term is used for values that are far from the bulk of the distribution of values for a given variable. This can be due to either data quality issues or rare (but true) events. Treating outliers generally requires a detailed understanding of the relevant application domain.



If we were to measure wind gusts of more than 40 m/s in a given year, we would classify this value as an extreme outlier; however, the question remains whether this value is due to a data error or if it is a real but rare measurement. Without expert knowledge of the domain (here: extreme weather situations), we would not be able to answer this question. If we removed this value, remaining data would, at first glance, look more sensible. In contrast, if we turn to a domain expert, we would learn that while high

wind gusts are rare, within a return interval of 50 years, wind speeds above 40 m/s are possible (Hofherr & Kunz, 2010). In this case, the specific measurement might still be a data error that needs to be investigated but might just as well be a valid measurement, and it would be wrong to remove it from the dataset.

Maximum Wind Speed in Germany



Hofherr & Kuntz (2010). Used with permission.

Summary

Data quality is one of the most important aspects when using data-driven decision models, machine learning, or artificial intelligence. Removing errors in the data as much as possible is of paramount importance since any remaining issues can potentially be learned by the algorithms during the training phase.

Dealing with low quality data, missing values, outliers, unexpected values, wrong formats, and other issues poses significant challenges when assessing the data quality and cleaning the data.

Exploiting further information or domain knowledge can help to determine whether a given issue is a data quality problem or, for example, a real but rare data point, as well as point to any systematic issues during data acquisition, processing, or storage.

Knowledge Check

Did you understand this unit?

You can check your understanding by completing the questions for this unit on the learning platform.

Good luck!

Unit 3

Feature Engineering

STUDY GOALS

On completion of this unit, you will have learned ...

... what feature engineering is and why it is important.

... how a wide range of encoding techniques works.

... how to build paired features.

... how to build features for data types such as structured data, images, or time series.

3. Feature Engineering

Introduction

The building of new features using an existing dataset is known as feature engineering. After all relevant data have been obtained, cleaned, and curated, it is often helpful to enrich these data with new information or explicitly create new variables based on this dataset that focus on a particular relationship or highlight some known relationship. These new variables are intended to help the later modeling or machine learning step to make better use of the available data. Feature engineering raises two important questions: Why should we do it if the data already contain the information implicitly, and what happens if the feature is "wrong"? The latter issue is addressed fairly easily. If the feature we build is "wrong" or not helpful for the project, then the feature selection process should identify and remove it from the list of available features.

Deep learning
This field is typically understood in the context of complex neural networks, which have many hidden layers between the input and output layer.

The question as to why create new features is more complex. The field of **deep learning** typically aims at avoiding this step and lets a complex machine learning algorithm extract all knowledge from the data. While this works well in many cases, adding additional information can still be helpful, for example, if we know from our domain knowledge that certain relationships are important or if the number of possible features is very large. In this case, factoring out this step can help to address which of the many features one might consider are statistically relevant for the problem. This avoids the inclusion of spurious correlations that are apparent in the training data but are not a characteristic of the physical system. Finally, deep learning approaches typically use large neural network-based architectures with millions of free parameters. If we only have a limited dataset available for training such an approach might not be feasible, but we can apply our expert knowledge to engineer new features and then use a comparatively small machine learning model to combine them optimally.

3.1 Building New Features

Structured Data

In many cases, we will work with structured data that can be represented by a table.

Structured Data				
Date	Product ID	Store ID	Promotion Flag	Sales Amount
2019-01-02	123	456	false	5
2019-01-02	125	456	false	3

Date	Product ID	Store ID	Promotion Flag	Sales Amount
2019-01-04	345	789	true	10
2019-01-07	378	978	false	7

These data often contain a reference to a **timestamp**. In the above example, events are listed by date. In other examples, such as sensor data, the specific event may also be identified by time and date with a precision relevant to the domain-specific applications, e.g., seconds, milliseconds, or even nanoseconds. Even though the information is —in principle—contained in the data, it is often useful to highlight different aspects of a timestamp explicitly using any of the following information:

Timestamp
A timestamp is an alpha-numeric value indicating the date and time of a specified event. Many databases and programming languages have dedicated types for storing and manipulating this kind of time information.

- month
- calendar week
- day of the week
- quarter
- season
- bank, religious, or school holidays
- "special days," e.g., religious or other events such as Easter, and Christmas
- the average time between specific types of events

We can also add more details about a specific variable and add these as new features. In the above example, the data are about the sales of a specific product (with a given product ID) in a specific store (with a store ID). We can augment our data by adding aggregate information, such as

- the peak (maximum) or average sales of a specific product,
- the ratio of average sales of a single product compared to similar products, or
- the average sales of all products in a specific store.

Combined with the features we can create from the timestamp, we can build many variations such as the increase of sales of a specific product on a given day of the week compared to the average sales. As an example, the Python package FeatureTools (FeatureLabs, n.d.) can automate many of these tasks. However, depending on the number of items and combinations, the number of newly created features grows very quickly.

Encoding Categories

Many variables are related to fixed categories; for example, a collection of fruit might contain the following types "apple, pear, plum, orange." Most modeling or machine learning approaches cannot deal with these categories directly but only with numerical values. Hence, any category needs to be transformed into a number.

This change can be done in several different ways. For example, we can assign a number to each category, e.g., “apple = 1, pear = 2.” In this way, each category is related to a specific numeric identifier. Categorical variables are identifiers that can be ordered, depending on their relationship. But the numeric values do not convey that one category is “better” than the other; “apples” are not intrinsically better than “pears” or vice versa. In contrast, when there are situations where a given category or identifier is preferable, we use ordinal variables. This is useful in situations such as mortgage applications: the category “low risk” is better than “medium” or even “high risk.” In this case, the assigned values 1, 2, 3 convey more meaning than simply denoting a specific category.

A more suitable way to encode categorical data without an ordinal relationship is called one hot encoding. In this scheme, a new variable is introduced for each possible category. In the example below, three new variables are needed to represent the colors “red, blue, green.” In order to represent a category, the corresponding newly created variable is set to “1,” the others to “0,” i.e., only one of the new variables is “hot.”

One Hot Encoding

Variable / Value	Red	Blue	Green
Red	1	0	0
Blue	0	1	0
Green	0	0	1

This approach works well when the number of categories is small. However, in the case of many categories, this approach does not scale well due to the number of additional variables required. We could first cluster the data and use the resulting cluster ID instead of the original variable. This way, the large number of variables is reduced to the few numbers of clusters found in the data. Another approach is target encoding (Micci-Barreca, 2001). The main idea here is to map the category to the expected value of the prediction target for this category. This way, the category is transformed to a quasi-continuous variable that can be used in the later modeling or machine learning algorithms.

In the case of ordinal variables, i.e., when the order of categories in the list matters, we can also apply sum encoding and Helmert encoding. In sum encoding, the mean of the target variable is compared to the mean of the target variable computed from all other values in the ordinal variable. In Helmert encoding, the mean of the target variable is compared to the mean of the target variable in all lower classes. Let's say we have the categories “cold, warm, very warm, hot, boiling hot,” and the specific event falls into the category “very warm.” Using sum encoding, we would compare the mean of the target

variable for “very warm” to the mean of the target variable considering the cases “cold, warm, hot, boiling hot.” If we apply Helmert encoding, we would instead compare the mean of the target variable for “very warm” to the cases “cold, warm.”

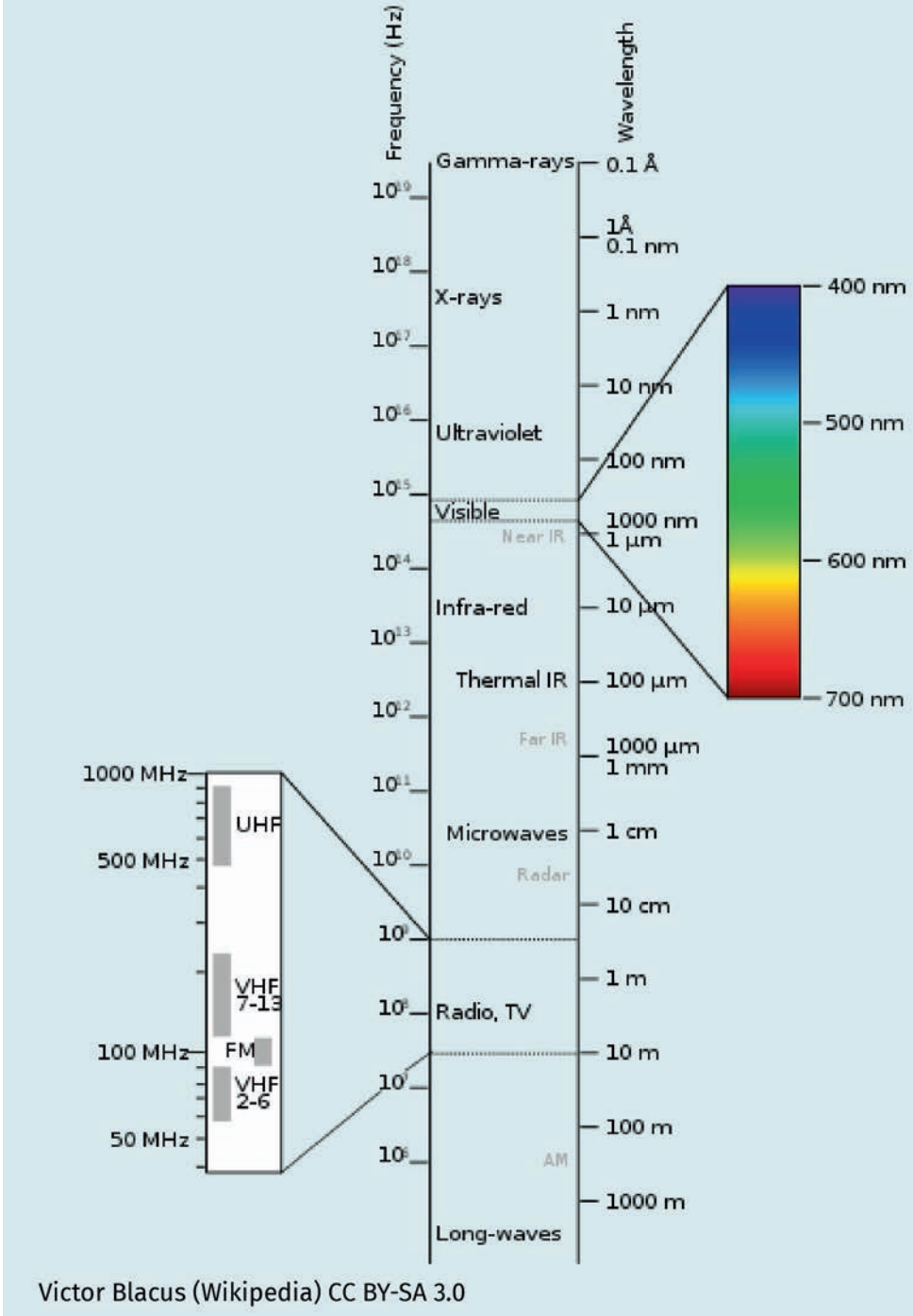
Pairwise Features

In many cases, correlation between features can contain more detailed information than the individual variables. For example, in the case of latitude and longitude, the joint information of the two can convey more meaning than considering each variable by itself. Depending on the model or machine learning algorithm used in the later modeling stage, the algorithm may be able to detect these two-dimensional dependencies by itself. However, even in these cases, it is beneficial to at least complement the list of feature variables with two-dimensional features. These pairwise interactions have the form $f(x_i, x_j)$ (Lou et al., 2013), where x_i and x_j are two feature variables and $f(.)$ denotes the functional form. For example, some modeling approaches will determine the term $f(x_i, x_j)$ as part of the training process. In others we have to train a submodel on the two feature variables such that $\hat{y} = f(x_i, x_j)$ is already an estimator of the target quantity built from these two features.

Images

Image data contain a wealth of information. Anything we can perceive visually can be encoded in an image. Images are also not limited to our natural perception using our (human) eyes. For example, we may choose to change the color or greyscale of an image to visualize details not visible or directly visible to the human eye. The electromagnetic spectrum (below) shows that the visual part of the spectrum is only a tiny part of the range of frequencies we can use.

Electromagnetic Spectrum



Victor Blacus (Wikipedia) CC BY-SA 3.0

This spectrum is exploited in X-ray images or **computer tomography** (CT), where the image is a representation of electromagnetic radiation outside our perception.

X-Ray Image

Computer tomography
This is a medical imaging technique where many X-ray images are taken at various angles (around a patient) in a given spot. These images allows us to reconstruct a 3D image of the interior, combining the X-ray images using a dedicated computer algorithm.

Infrared images show details such as the heat distribution of people or objects.

Infrared Image

Using images hence does not necessarily mean that the images show the same things we would see with our biological eyes.

Besides using the raw image “as is” as input to a model or deep learning algorithm, it is often useful to add additional information to the images. One approach is called semantic image segmentation where those parts of images are identified that belong together. This allows us to identify which kind of objects are present in an image and where, as well as extract additional metainformation about these images where the details about the identified objects are added in as structured data. For example, applications in autonomous driving requires not only identifying which objects are in a picture but also judging how the car should respond. In the photograph below, we would have to identify other cars, cyclists, pedestrians, as well as static objects such as trees, houses, or—most importantly—where the road goes, how the lines are divided, and which traffic signs or traffic lights can be seen. Crucially, some types of objects are more important than others. While it is indeed mandatory to avoid driving into buildings or urban lighting, autonomous cars need to be acutely aware of pedestrians and moving objects. In this respect adults are, at least in general, more predictable as they understand the challenges and dangers of moving traffic better than a small child chasing after a ball. Identifying all relevant objects enables the car to participate safely in traffic. Performing this kind of segmentation repeatedly on a sequence of images also allows us to extract movement patterns, for example, if a child is indeed running after a ball.

Urban Traffic

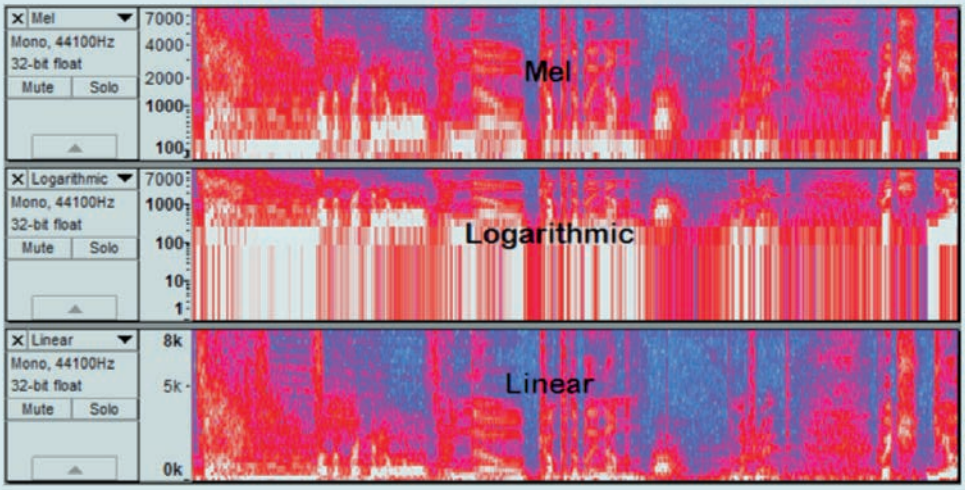
Further information can be added to the images by adding depth information. This can be done in several ways. Data from external sensors such as **LIDAR** can be combined with the segmented image data, adding the extra information to the metadata extracted from the image (e.g., the car ahead in the lane is 20 meters in front). Alternatively,

LIDAR
In "LIght Detection And Ranging" (LIDAR), a pulsed light originating from a laser is sent to various points in the surrounding environment, and the response in the reflection pattern is used to create a map of where and how far away objects are.

depth information can be estimated from single images using dedicated models (Alhashim & Wonka, 2018), and the color scheme of the image can be used to represent this information.

Images can also be used to represent information that has no direct connection to what we would perceive visually. An example is the Mel Spectrogram developed by Mermelstein (Chen, 1976; Davis & Mermelstein, 1980) building on an idea by Bridle and Brown (1974). The main idea is that an audio signal is shown as an image, and the time is represented by the x-axis. The y-axis represents the frequencies, and a color is used to indicate how strongly a given frequency is present at any given time. Instead of being limited to the frequency, the mel scale (Stevens et al., 1937) is adapted to how we perceive audio frequencies as humans. The image below demonstrates the same signal three times using a linear, logarithmic, and the mel scale. These images can be used to represent and analyze audio signals in the same way images showing something we would perceive visually as well, since a deep learning model aims to extract the underlying information, whether or not we as humans would do it in the same way.

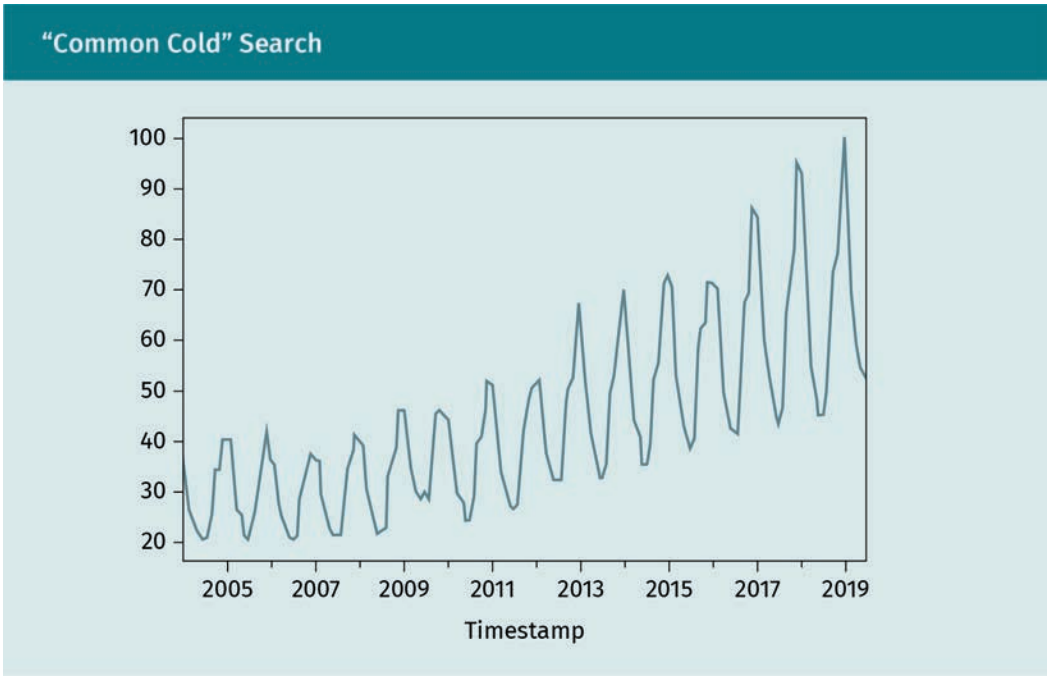
Frequency Spectrum



Jacynycz (2017). CC BY-SA 4.0.

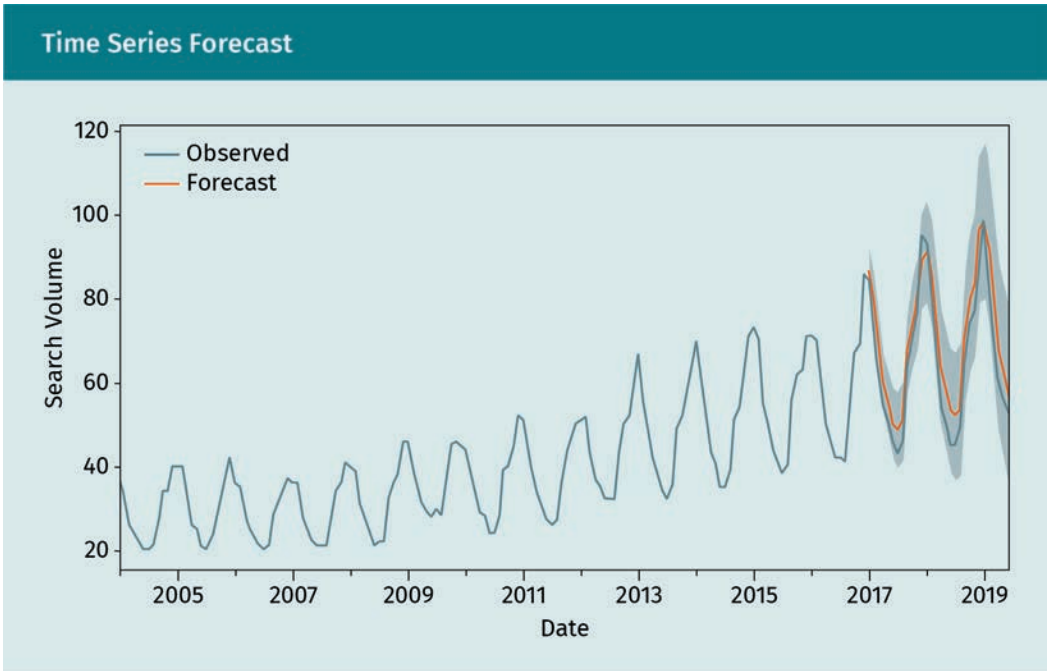
Time Series

Time series data are a special kind of structured data where the values of a given variable are recorded at various moments in time. This means that we have repeated measurements of a single variable, ordered by when they were taken. These kind of data are pervasive and found in numerous applications and scenarios from supermarket sales records or the number of sun spots throughout the years to the search frequency for "common cold" on a search engine on the World Wide Web.



The graph shows a typical behavior of time series data: the same variable is observed repeatedly and (typically) a periodic pattern is observed, in this case related to the seasons. Additionally, the overall volume increases with time, i.e., the time series includes a rising trend.

In many cases, we will analyze time series data, as well as use them in an application. In time series classification, we are concerned with using the time series data to decide whether, e.g., an event will fall in one or the other category. For example, we may measure various parts of a machine (e.g., a time series of sensor data) to decide if the machine is likely to break soon and needs maintenance. In time series regression, we are focusing on predicting the future values of a time series, as illustrated below. Anticipating future search patterns might allow pharmacists to anticipate the next wave of the common cold and stock up on relevant medicine. In this example, the last part of the data have been omitted from the prediction as an independent test sample. When building a sophisticated model, it is often helpful to include other modeling approaches as well. This is particularly useful with machine learning approaches, such as neural networks, as these can combine different information optimally. In this case, the later model can combine both the time series data (either the original one or the one split by trend, season, and residual) as well as first estimates of future values of the time series data. The simplest forecast would be to repeat the last data point in the time series. A little better is to use the moving or rolling mean (Booth et al., 2006), which treats all past values with the same weight or the exponentially weighted moving average where recent occurrences have a larger weight than those that are further in the past. More sophisticated approaches include ARIMA models (Box & Jenkins, 1970) or the approaches by Holt (2004) and Winters (1960).



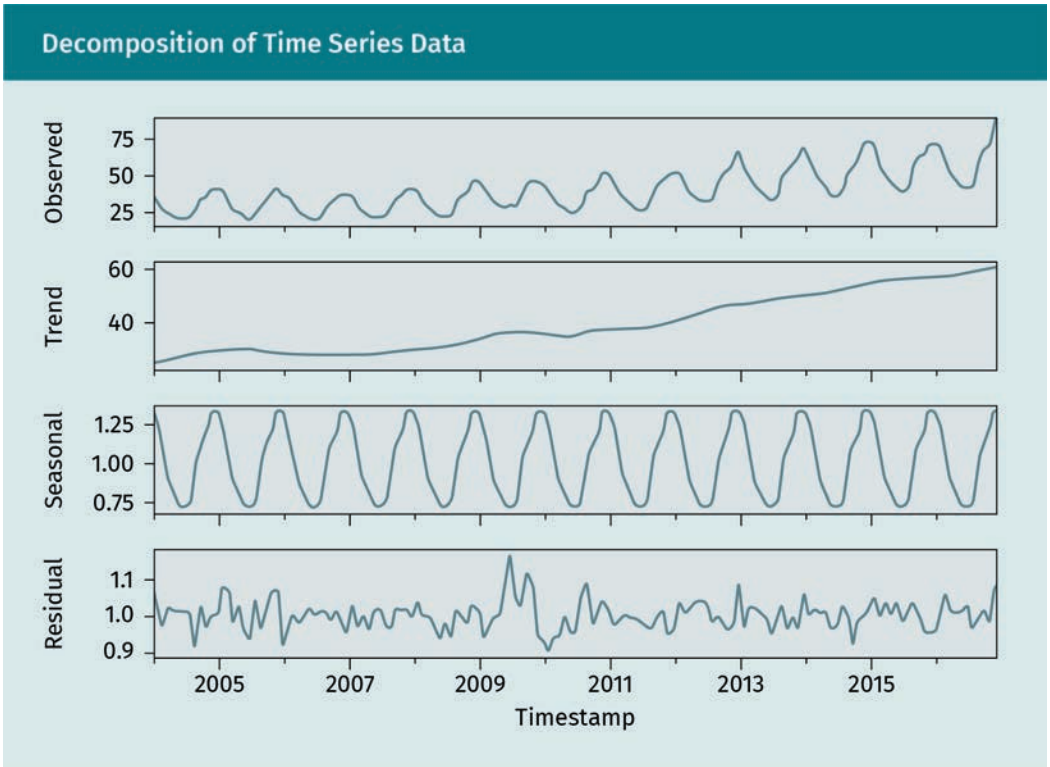
In addition, time series data can be characterized by a wide range of metrics, quite similar to descriptive statistics in the case of probability distributions and histograms. For example, we can characterize a time series by its mean or median value, its variance, the minimal or maximal value, the value of the peaks in the signal, or the number of peaks. Since the data cover some length of time during which they were acquired, we can also divide the signal into separate windows and determine the characteristics per time window. For example, the mean value per minute or the number of peaks per second might be relevant to our task. Since a wide range of potential features can be obtained from any number of intervals from the time series data, characterizing time series data in this way typically leads to a large number of variables which would enter the final model for either time series classification or regression. Beyond the practical aspects of handling this large number of feature variables, the more variables are used in the model, the higher the risk that the model picks up spurious correlations. Hence, the features need to be reduced again such that only the relevant features remain (Christ et al., 2018).

3.2 Splitting Variables

When using a variable in a model or machine learning approach, we assume that all relevant information is already encoded in this variable and can be extracted by the model. However, in many cases it is helpful to split variables into components to make this information more explicit. Instead of one variable conveying a wide range of different aspects of information, separate variables allow us to focus on one particular aspect at a time. When using time series data, it is often useful to separate the components, as shown below. The top graph shows the time series data “as is,” i.e., as measured. We observe an increase in the overall volume as well as a seasonal pattern. Split-

ting the original data into trend, **seasonal variation**, and residual focuses on the overall behavior (trend), the cyclic variations (season), and the short-term fluctuations separately.

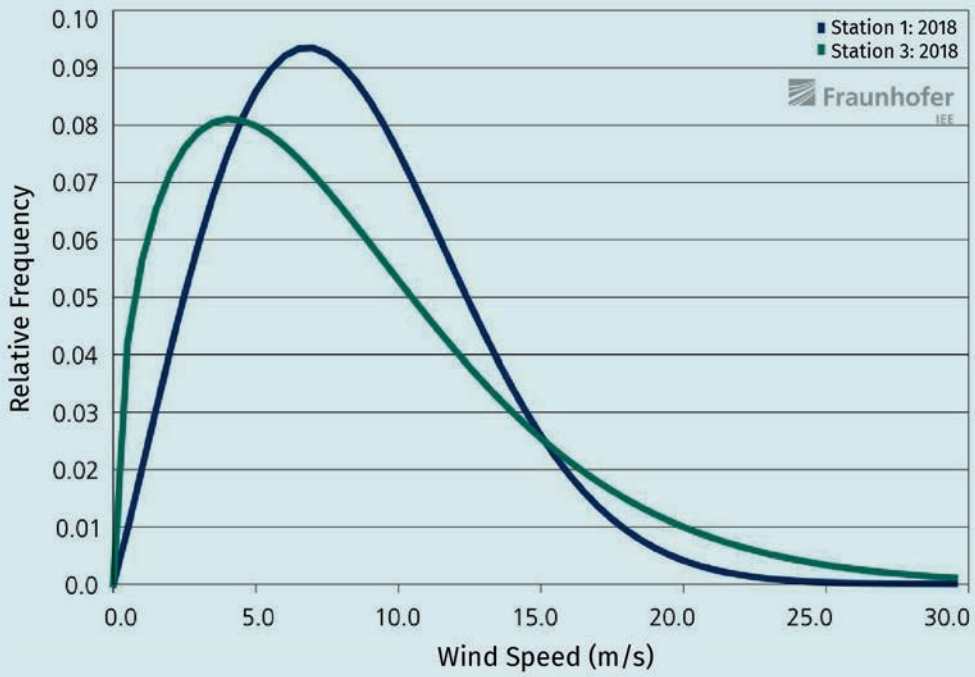
Seasonal variation
Many time-dependent data show a regular pattern that depends on the time the signal was recorded. For example, temperatures are generally higher in summer than in winter.



In some cases, variables contain outliers. First, outliers need to be identified with tools like PyOD (Zhao et al., 2019). Next, we have to identify whether the outlier is a real and valid measurement or needs to be addressed as a data quality issue. If the outlier does represent a true and valid data point, then this outlier needs to be treated carefully. We need to include it into the training data and the model since it is a valid measurement; however, it may cause issues in the modeling phase. For example, regularization schemes may remove the value during the modeling phase, or the outlier may influence the overall model too much and skew all other predictions.

Consider the wind speeds measured at the shore of Germany. In 2018, a wide range of wind speeds was measured between zero and 30 m/s. Now we imagine that a wind speed of 40 m/s was measured on one day. After making sure that the sensors were working correctly, we can assume that this value represents genuine data, even though it is rare, as the map below illustrates.

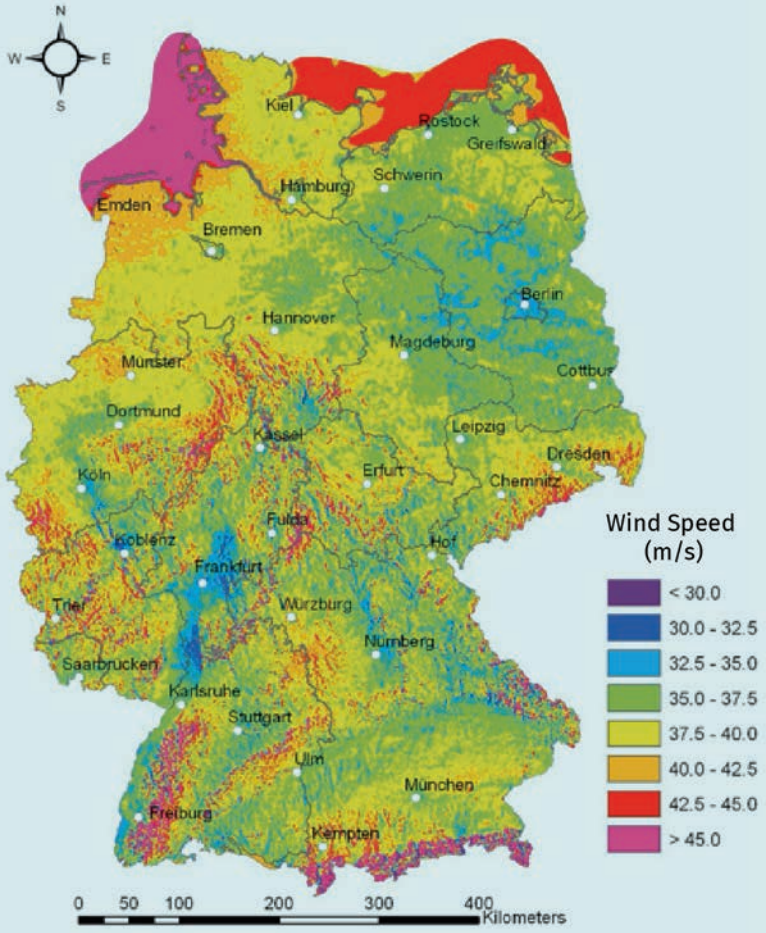
Distribution of Coastal Wind Speeds in Germany



Fraunhofer IEE (2020). Used with permission.

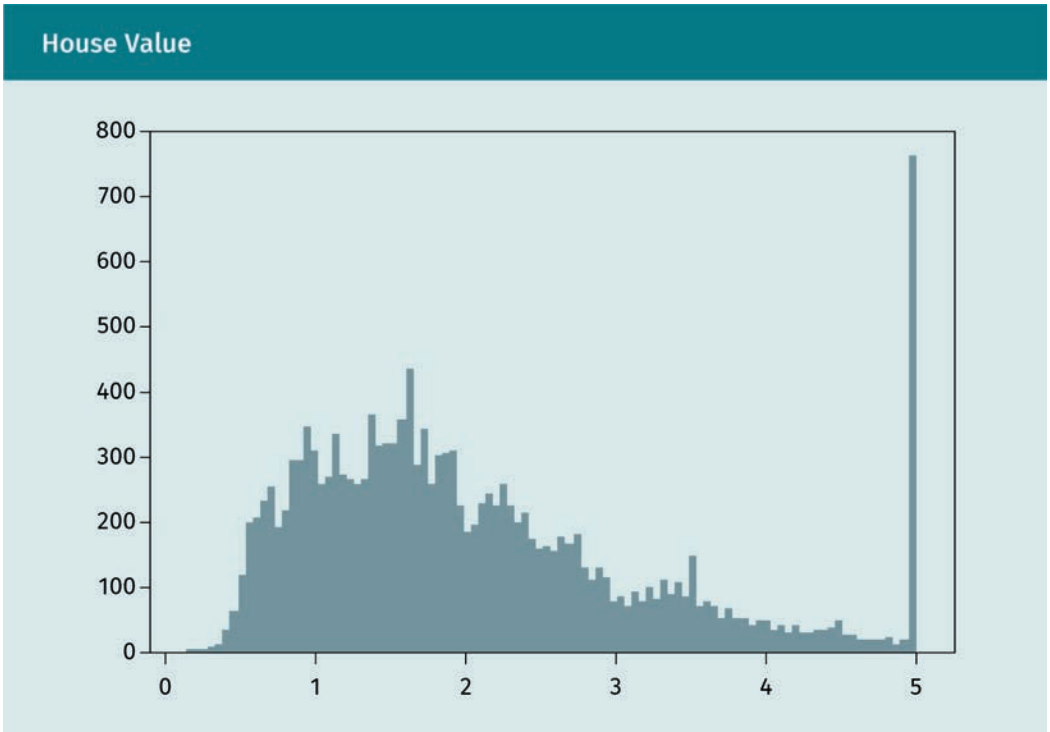
While the event is important—at least in the sense that it corresponds to exceptional circumstances—just adding a single or even a few of these measurements to the bulk of the data will not improve the model much. Either these few events are removed using some regularization scheme, or the overall model performance might degrade. It is therefore better to split the variable into two ranges: "normal wind" and "exceptionally strong wind." Depending on the modeling approach, we are either able to include these variables in our model directly, or we need to build a first model identifying the strong wind and then work with the "normal wind" distribution as before.

Maximum Wind Speed in Germany



Hofherr & Kuntz (2010). Used with permission.

The same approach should be taken if one of the variables exhibits several distinct behaviors such as house prices (see the graph below) in the California Housing dataset (Pace & Barry, 1997). In this example, the target variable shows a continuum of property prices on an adapted scale between zero and five and a distinct peak at the maximum value, representing a cutoff. Most modeling or machine learning approaches will struggle to predict such a distribution correctly. Instead, we can split the variable into a binary variable (normal/very expensive) as well as the continuum. Depending on the capabilities of the models used, we can either use the split variable directly or will have to build separate models. In the latter, we could build a sequence of classification to distinguish between normal and very expensive property prices, followed by a regression model in the case of “normal” property prices.



3.3 Feature Engineering Exploiting Domain Knowledge

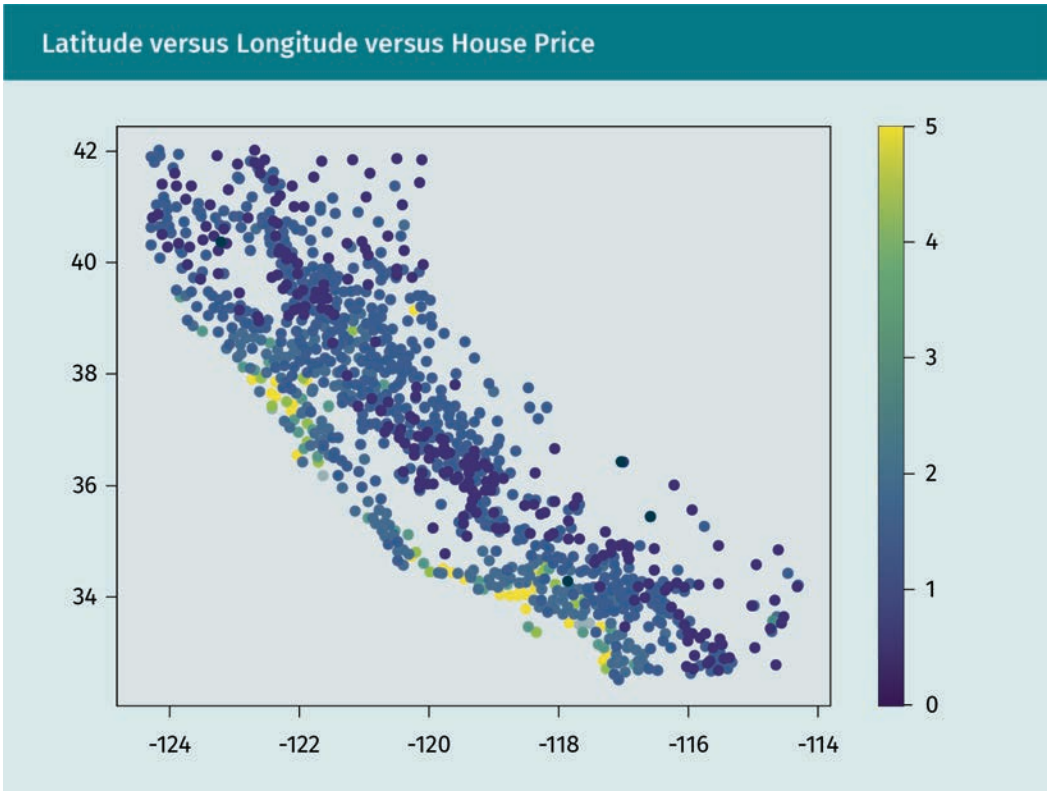
Application domain
The application domain specifies the context in which the data were recorded or the predictions and decisions to be used. Examples for application domains are automotive, retail, insurance, and stock markets.

In many cases we can exploit our expert knowledge about a specific **application domain** to combine features into more meaningful variables. As experts, we might know that the sum, difference, or ratio of two variables is more useful than the "raw" variables themselves. Adding the known relationship explicitly allows the model or machine learning algorithm (later in the processing pipeline) to make better use of the information. If we suspect that the combined variable might not capture all details, we can add it to the raw variables. In this case, the later modeling phase can either make use of the combined variables (and the raw variables are removed from the list of all variables using defined feature selection criteria), or the model can learn when our combined variable adds useful information—and when it is better to rely on the raw variables.

Beyond simple combinations, we can also use a first model or machine learning algorithm to exploit a known relationship before feeding all information into the main model or algorithm. This can be illustrated with the "California Housing" dataset which predicts house prices based on a range of feature variables (Pace & Barry, 1997). Amongst others, the list of available features includes the latitude and longitude of the properties. Intuitively, we know how to exploit this information best since the variables "latitude" and "longitude" are known from everyday life. The graph below shows the price of the properties (on a transformed scale between zero and five) arranged by latitude and longitude. To our expert knowledge, this visualization confirms our expectation that the highest prices are in densely populated and popular areas such as San

Francisco, the Bay Area, or Los Angeles. Prices are lower outside these areas and (almost) no properties can be found in the deserts. However, a model or machine learning algorithm does not have our domain or background knowledge. It would not "know" that the features "latitude" and "longitude" are best combined in a specific way and that the variables have a more detailed meaning beyond the numbers, as they refer to specific locations in California. To exploit the information optimally, we can build a specific small model just including these features, or we can use an initial machine learning algorithm. In our example, a regressor based on k-nearest neighbor clustering is a relatively simple approach to combining latitude and longitude into a prediction for our target "house price," as we expect that high-value properties will be close to each other, and the same for lower value properties.

This means that our full model later in the modeling process not only include features found in the dataset but also the output of small "intermediate" estimators for our target variable in a hierarchical approach.



Summary

Feature engineering is the process of creating new variables from the “raw” data with the aim of making information more accessible or explicit. This allows the later modeling stage to make better use of the data. Depending on the type of data, a wide range of new features can be built using different techniques. In some cases, it is also helpful to split variables into individual components that focus on specific aspects of the recorded data.

New features benefit from including external or expert knowledge into our modeling approach. The advantage is that known relationships don’t need to be rediscovered first; however, the model risks learning the wrong relationships if the domain knowledge is not correct.

Knowledge Check

Did you understand this unit?

You can check your understanding by completing the questions for this unit on the learning platform.

Good luck!

Unit 4

Feature Selection

STUDY GOALS

On completion of this unit, you will have learned ...

... why feature selection is important.

... about the false discovery rate and how it relates to feature selection.

... how to use methods for univariate feature selection.

... how to employ models in model-based feature selection.

4. Feature Selection

Introduction

Features
These are variables used to build a model or train a machine learning algorithm. Features are often also called just variables or independent variables.

Selecting **features** is one of the most important steps in building a good model. After all data have been cleaned and additional features have been built with feature engineering as required, we decide which of the features are relevant for the task and should be included in the following modeling stage. Naively, we might be tempted to add all available features to the model. However, this approach is not useful in most cases. First, many features may be highly correlated with each other. This means that most of the information is contained in one of the variables, and adding further variables increases the amount of information only a little bit, while adding a considerable amount of noise—depending on the concrete task and dataset— may do more harm than good. Although each independent piece of information should help to improve the model, the additional noise may make it more difficult for the model to converge to an optimal setting; hence, any advantage from the additional information may be destroyed by adding too much noise. Adding many features increases the risk of adding spurious relationships that are an artifact of the data available but not of the system we want to analyze. Adding a new feature to our list of variables corresponds to the question: "Are these data relevant for the prediction we need to make?" The more we ask this, the likelier that the answer is "yes" due to some statistical fluctuations in the data rather than that there is a relationship to be learned from the data. It is preferable from a practical point of view to reduce the number of features to a minimum, both to make handling the data involved easier and to improve the convergence behavior of the model.

Hence, we aim to use as many features as needed and are justifiable for a given amount of data, while reducing the complexity as much as we can.

Controlling for the False Discovery Rate

We now formalize our intuition discussed above that the more features we have, the more likely we are to accept a spurious connection between a feature and our target value. Adding a feature to the list of variables we include in our model is equivalent to asking the question: "Is this variable important for our task?" which we can interpret as many such hypotheses being tested at the same time. Hence, we have H (the hypothesis to be tested, i.e., the feature is not relevant (the null hypothesis)), m (the number of hypotheses we want to test), and T/F (true / false) for the result of the test. Since we don't know if the hypothesis is really true or not, we can only observe if we perceive the hypothesis to be true given the current data. We denote this as D (discovery) or N (non-discovery). Our observables are then m, D, N. Note that T and F are not directly observable. The table below summarizes this.

Observables			
	H accepted	H rejected	Total
H True	TN	FD	T_0
H False	FN	TD	T_1
Total	N	D	m

We note immediately that FD (False Discoveries) are bad for our model since we accept features into our final list of features that only contain spurious connections with our target variable. Hence, if we have many features, we need to control the False Discovery Rate (FDR) at which new features are "discovered" to be important and included in our model. Given a threshold α, the Benjamini-Hochberg-Yekutieli procedure (Benjamini & Yekutiel, 2001; Benjamini & Hochberg, 1995) controls the FDR

$$\mathrm{FDR} = \mathrm{E}\left[\frac{\mathrm{FD}}{\mathrm{FD}+\mathrm{TD}}\right] = \mathrm{E}\left[\frac{\mathrm{FD}}{\mathrm{D}}\right] < \alpha$$

where $\mathrm{E}[.]$ is the expectation value which we need here as TD and FD are (unknown) unobserved random variables. The FDR quantifies the intuitive assessment that four false discoveries out of ten rejected null hypotheses are worse than 20 false discoveries out of 100 rejected null hypotheses. The Benjamini-Hochberg procedure (Benjamini & Hochberg, 1995) for uncorrelated hypotheses works in the following way:

1. Test the null hypotheses H_1, H_2, H_3, ..., H_m (i.e., the feature variable is not correlated to the target variable). Each of these tests comes with a **p-value**: p_1, p_2, p_3, ..., p_m.
2. Now order the p-values by ascending value, denoted by $\mathrm{P}(\mathrm{i})$, i.e., $\mathrm{P}_{(1)} \leq \mathrm{P}_{(2)} \leq \ldots \leq \mathrm{P}_{(\mathrm{m})}$
3. For any threshold value α, find the largest k such that $\mathrm{P}_{(\mathrm{k})} \leq \frac{\mathrm{k}}{\mathrm{m}}\alpha$
4. Reject all null hypothesis (i.e., accept the features, declare a discovery) for all $\mathrm{H}_{(\mathrm{i})}$ with $\mathrm{i} = 1, \ldots, \mathrm{k}$

p-value
The p-value is a statistical measure to determine whether an effect is significant or spurious. Typically, effects for which p > 0.05 are deemed significant.

This approach controls the FDR at level α, and the threshold value α is chosen such that the number of rejections is maximized. In case of correlations, the Benjamini-Hochberg-Yekutieli procedure (Benjamini & Yekutieli, 2001) has to be used where $\mathrm{P}_{(\mathrm{k})} \leq \frac{\mathrm{k}}{\mathrm{m}\cdot \mathrm{c}(\mathrm{m})}\alpha$ and $\mathrm{c}(\mathrm{m}) = \sum_{\mathrm{i}=1}^{\mathrm{m}} \frac{1}{\mathrm{i}}$.

4.1 Univariate Feature Selection

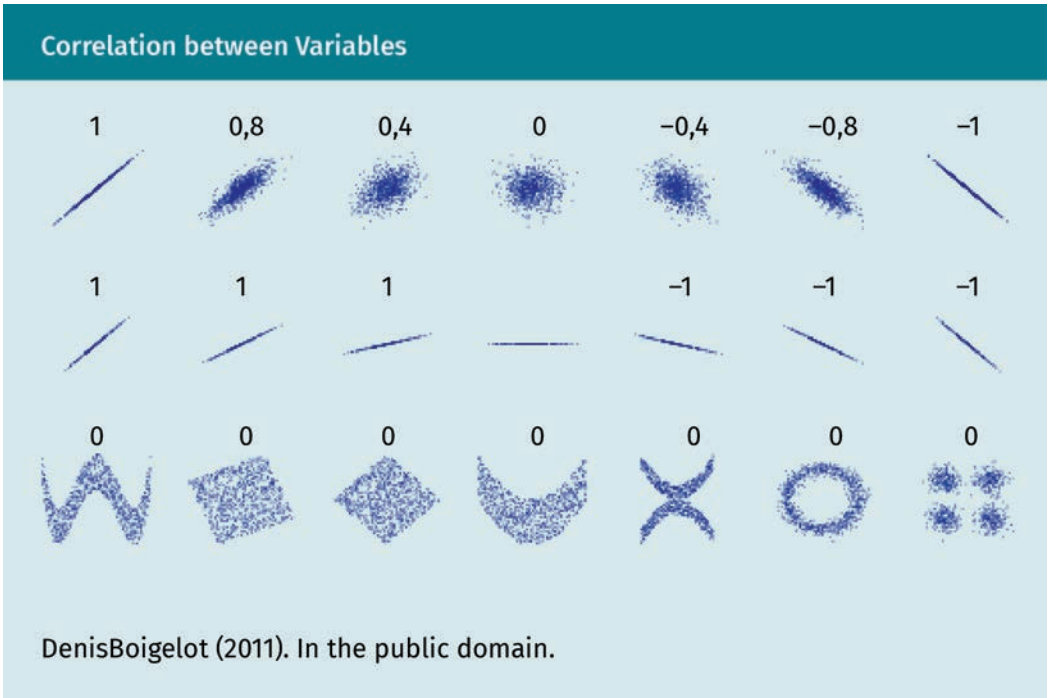
Univariate feature selection focuses on methods that only use the feature variables and the target of the model. This has the advantage that the evaluations are valid for any modeling approach we might want to use at a later stage. However, these approaches are often less powerful compared to model-based feature selection.

Analyzing Correlations

The first step in analyzing the behavior of the feature variables is to look at the correlation between the variables, where the Pearson correlation coefficient (Pearson, 1895) is defined as

$$\rho_{X,\,Y} = \frac{\mathrm{cov}(X,\,Y)}{\sigma_X \sigma_Y}$$

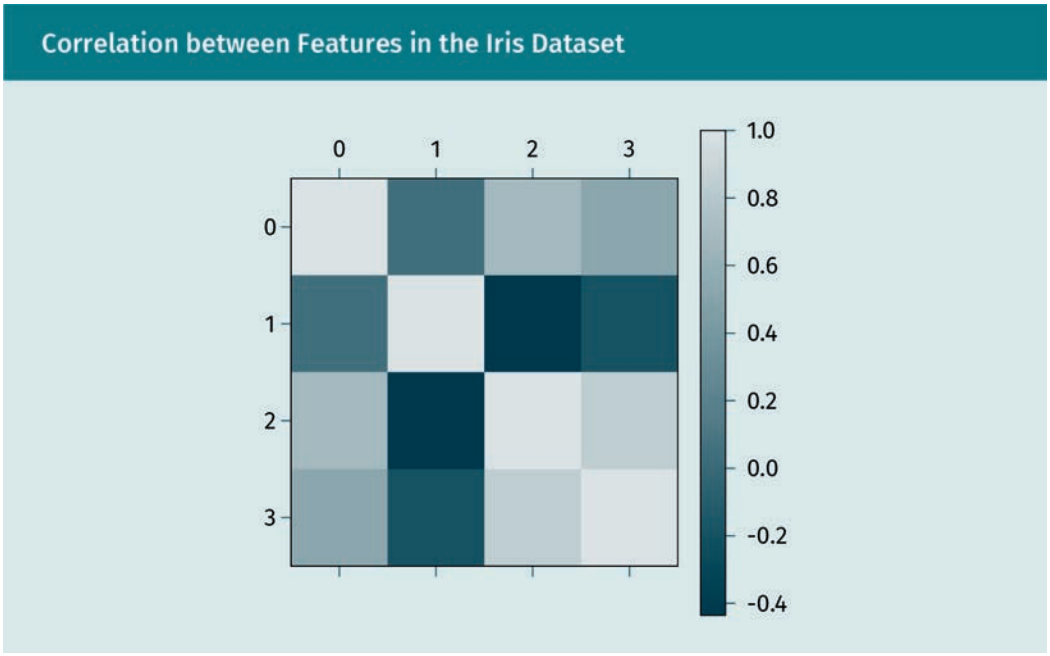
where $\mathrm{cov}(X,\,Y)$ is the covariance between the variables, and σ_x and σ_y are the standard deviations of feature variable X and Y. The correlation coefficient is in the range of —1,..., +1 where —1 indicates a negative total linear correlation (i.e., the variables change in the same way if one of them is changed), 0 means that there is no linear correlation, and +1 indicates positive total linear correlation. Note that the Pearson correlation coefficient is only sensitive to linear correlations; even with $\rho = 0$, the variables may depend on each other as illustrated below.



A variant of the Pearson correlation coefficient is the Spearman's or rank correlation coefficient. Instead of calculating the correlation coefficient using the values of the variables themselves, as in the Pearson correlation coefficient, we use the rank of the variables. The rank is the relative position of the respective data value within all data values, i.e., the first value (with a given numeric value), the second value (with another numeric value), etc. This means we compare the way they are ordered within the dataset and not the values themselves. This is useful if the data are very unevenly distributed or have a nonlinear relationship.

The (Pearson) correlation between feature variables is illustrated below using the Iris Dataset (Fisher, 1936). This dataset is concerned with identifying to which of the three types of iris flowers a given flower belongs, depending on the variables

- 0. Sepal length
- 1. Sepal width
- 2. Petal length
- 3. Petal width



As we can see from the color, features 2 (petal length) and 3 (petal width) are highly correlated with each other. We expect that only limited independent information is added by including both variables in the later model. When handling large numbers of features, we often encounter blocks of variables that are highly correlated with each other. While in many cases we wouldn't necessarily remove these variables in a first step, the correlation analysis is a useful first step to further investigate the relation between variables.

Feature Significance

Calculating the feature significance is a univariate selection method exploiting the correlation between feature variables and the target variable. First, we diagonalize the correlation matrix which has the benefit that all transformed variables are linearly independent from each other. Then, the total correlation of all new (transformed) feature variables with respect to the target variable are given by

$$\rho_{\mathrm{T}}^2 = \sum_{\mathrm{i}=1}^{\mathrm{N}} \rho_{\mathrm{T,i}}^2$$

where $\rho_{\mathrm{T,i}}$ is the correlation coefficient between the target variable (T) and the i^{th} transformed variable. We can then calculate the loss of the total correlation if one of the variables $\mathrm{x_j}$ is removed

$$\rho_{\mathrm{x_j}}^2 = \rho_{\mathrm{T}}^2 - \rho_{\mathrm{T},\mathrm{x_1},\mathrm{x_2},...,\mathrm{x_{j-1}},\mathrm{x_{j+1}},...,\mathrm{x_n}}^2$$

The feature significance of this variable is then defined as the correlation multiplied by the square root of the number of data samples $(\mathrm{n_{data}})$

$$\mathrm{S_j} = \rho_{\mathrm{x_j}}\sqrt{\mathrm{n_{data}}}$$

Calculating this metric iteratively for each feature variable helps identify those which are most important when building a model and sets a suitable threshold to reject the others.

χ^2 Feature Selection

This approach is used for classification tasks with categorical features. For discrete events, the χ^2 test statistics are given by

$$\chi^2 = \frac{\left(\mathrm{n_i} - \mathrm{n_{t,i}}\right)^2}{\mathrm{n_{t,i}}}$$

where $\mathrm{n_i}$ is the number of observed events and $\mathrm{n_{t,i}}$ is the number of expected events. To use this for feature selection, we want to test whether a given feature is independent of the target variable or not. We recall that two variables are independent if $\mathrm{P(A\cap B)} = \mathrm{P(A)P(B)}$. In order to find the expected values (under the hypothesis that the variables A and B are independent), we calculate the corresponding probabilities $\mathrm{P(A)}$ and $\mathrm{P(B)}$ from the contingency table (Pearson, 1894).

For example, the contingency table for a feature might look like this:

Contingency Table			
	Label = True	Label = False	Total
Variable = True	40	10	50
Variable = False	30	20	50
Total	70	30	100

In this case, the expected value of cases where both the feature variable and the label are true is $E_1 = n \cdot p(Var = True) \cdot p(Label = True) = 100 \cdot (70/100) \cdot (50/100)$ and, correspondingly, for all other combinations. These values are then compared to the observed values. In the case of high χ^2 scores, the null hypothesis, i.e., that the (categorical) feature variable and the target variable (label) are independent, should be rejected. This means that the variable is important for the model and should be kept.

The method can also be used on numeric data where the continuous variables are discretized appropriately (Liu & Setiono, 1995).

4.2 Model-Based Feature Selection

Model-based feature selection makes use of an intermediate model to determine which features are relevant. This model does not have to be the same as the final modeling approach. However, in practice, both the model used to select the features and the final approach tend to be the same or very similar because features selected and ranked as highly important by one modeling method may not be as useful for a different modeling method.

Feature Selection Using the L1 Norm

This approach uses a linear model which applies a penalty based on the L1 **norm** (Tibshirani, 1996). This approach adds a term $\lambda \sum_{i=1}^{n} |a_i|$ to the model where λ determines the strength of the regularization and a_i are the free parameters of the model. For example, a simple linear regression model expresses the target variable as $y = \sum_i a_i x_i$; hence, the regularized model would be $y = \sum_i a_i x_i + \lambda \sum_i |a_i|$. The L1 norm-based regularization has the advantage that non-significant contributions are forced to zero which can be exploited to select only the relevant features (Tibshirani, 1997; Baraniuk, 2007).

Norm
The norm is a term from linear algebra that assigns a positive size or length to a vector. This can be done in several ways leading to different definitions of various norms.

Feature Selection Using Random Forests

Random forests is an ensemble method that uses many decision trees. We can exploit one of the key characteristics of decision trees to identify the most important feature variables. By construction, decision trees build the nodes of the tree by splitting the variables into two or more regions. This split is done according to a metric used internally in the decision tree. For classification tasks the Gini impurity is typically used, whereas regression tasks use the residual sum of squares. In either approach, the split is done such that it increases the amount of order or reduces the variance. As the training over the ensemble of decision trees progresses, variables and splits that increase the order or reduce the variance the most tend to happen at the higher nodes in the decision trees. This allows the selection of the most important features by removing the nodes below a given threshold.

Feature Importance

Once we have settled on a modeling approach, we can determine the importance of the available features using permutation feature importance for labeled data, i.e., for a classification or regression task where we know the true values. This concept was first suggested in Breiman (2001) for random forests and later generalized by Fischer et al. (2018). This method uses an iterative approach to determine the importance of each feature:

1. Use the chosen modeling approach with all available features and establish a baseline. Using a specified error metric such as the mean absolute deviation (MAD, $\frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$) or mean squared error (MSE, $\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$), calculate the model error.
2. For each feature variable x_j ,
    1. Permutate (or shuffle) all values of x_j. This breaks any relationship between the behavior of the variable x_j and the target variable (label).
    2. Train the model.
    3. Calculate the new value of the error metric with the shuffled feature variable x_j .
    4. Compare the baseline value and the new value of the error metric, either as a ratio or difference between the two values.

The features variable x_j can then be sorted according to the ratio or difference between the baseline value of the metric and the value obtained after shuffling the values. If the metric doesn't change much, the feature is not very important for the model and can be removed.

Although this approach is quite intuitive, correlated variables are challenging in this approach. This can be both a strength and a weakness. It can be a strength if we want to determine how much a model relies on one of the variables and how much independent information is added by the other. However, if two variables are highly correlated, one of them will necessarily be flagged as "unimportant" as most of the information is already contained in the other. Depending on the concrete problem, such a

result may be unexpected as it may run counter to our intuitive understanding of the problem or our expert and domain knowledge. In these instances, we need to keep in mind that this effect is mainly due to how correlations are considered in this approach.

As a variant, we can also iteratively remove the feature altogether and observe the change in the metric. This approach is also called recursive feature elimination (RFE).

Shapley Values

Features can also be selected (Cohen et al., 2005; Zaeri-Amirani et al., 2018) using Shapley values (Shapley, 1953), which originate in game theory. It is based on the concept of "coalitional games" in which a number of players achieve a given result. The Shapley value is the metric assigned to the players according to their contribution towards the final outcome. In the case of feature selection, the "players" are the feature variables and the outcome is the value of the error metric such as the MAD or MSE of the supervised training of a given model. In this case, the Shapley value establishes a relationship between how much each feature variable contributes to the outcome, i.e., to the value of the error metric. Unlike other approaches the Shapley values take into account that feature variables "cooperate" with each other, i.e., they don't work in isolation. However, this implies that all possible combinations of features have to be considered, which is typically computationally prohibitively expensive in case of many features.

Summary

Feature selection is a key part of building models. Univariate feature selection exploits the relationship between features and the target. It can be used with any later modeling approach. Model-based feature selection uses an intermediate model to determine which features are relevant for the task. Since many features can be present in the original data or constructed with feature engineering, it is important to confirm that features are relevant for the task, avoiding spurious connections between the features and the target variables. Controlling for the false discovery rate (FDR) can help accomplish these goals.

Knowledge Check

Did you understand this unit?

You can check your understanding by completing the questions for this unit on the learning platform.

Good luck!

Unit 5

Building a Predictive Model

STUDY GOALS

On completion of this unit, you will have learned ...

- ... how to build and improve models using a hierarchical approach, ensemble models, or boosting.
- ... why interpreting the output of a model as a probability or probability distribution is important.
- ... the importance of developing interpretable models in a wide range of applications.
- ... how to build interpretable models.

5. Building a Predictive Model

Introduction

Building a predictive model is the central part of generating value with data where new or unknown data are used to infer estimates for the relevant task. For example, a Web shop or bank might want to know whether a given transaction is fraudulent, companies are interested in whether a given customer has a high churn risk, or supermarkets need to estimate future demand for their products in order to estimate how many products are needed from the supply chain.

Before starting to build a predictive model, it is imperative that we have completed the following steps:

- Use case. We need to have a clear understanding of the target variable, i.e., the quantity we need to predict, whether we build a model for a classification task or for the estimation of a quantity (regression).
- Evaluation. After defining the use case, we need to have a good understanding of how the prediction is used in the context of the relevant use case and how the predictions will be used, e.g., to make data-driven decisions or gain insights into a specific problem. This allows us to define how we are going to evaluate the predictions made by the model. We should keep in mind that in practical applications, the impact of the model is often more relevant than technical metrics. We should, of course, strive to build a good model; however, a small improvement in some technical metric does not necessarily translate into a valuable increase on the impact of the model. From a practical or business point of view, there is often a point where a model can be deemed as "good enough," i.e., the objective of the use case can be fulfilled and a small increment in performance does not warrant the cost associated with development time, adding further data sources, etc. Arguably, the lesson of learning when to stop is one of the hardest for any data scientist working in industry.
- Data acquisition, storage, and processing. Before we can begin to build the model, all relevant data have to be acquired. During the modeling process we may, of course, realize that additional sources of data are required to optimize the model. However, when we start to build the model, a reasonable dataset should be available. Moreover, we need to be able to access and process the data easily—while we're not yet looking at building production-ready pipelines at this stage, any encumbrance in accessing the data will slow down the model development process and distract us from building the model.
- Data quality. Similarly, the data we wish to use to build a model should be cleaned as much as possible beforehand. This does not exclude that we may find more data quality issues as our understanding of the data progresses, but all issues we have initially found should be addressed before building the model.
- Data exploration. We should have a good understanding of the data and the relationships between the various feature variables before building our model. Although our understanding of the data deepens considerably during the modeling phase, we should have a good understanding beforehand, especially about how the data are connected to the use case.

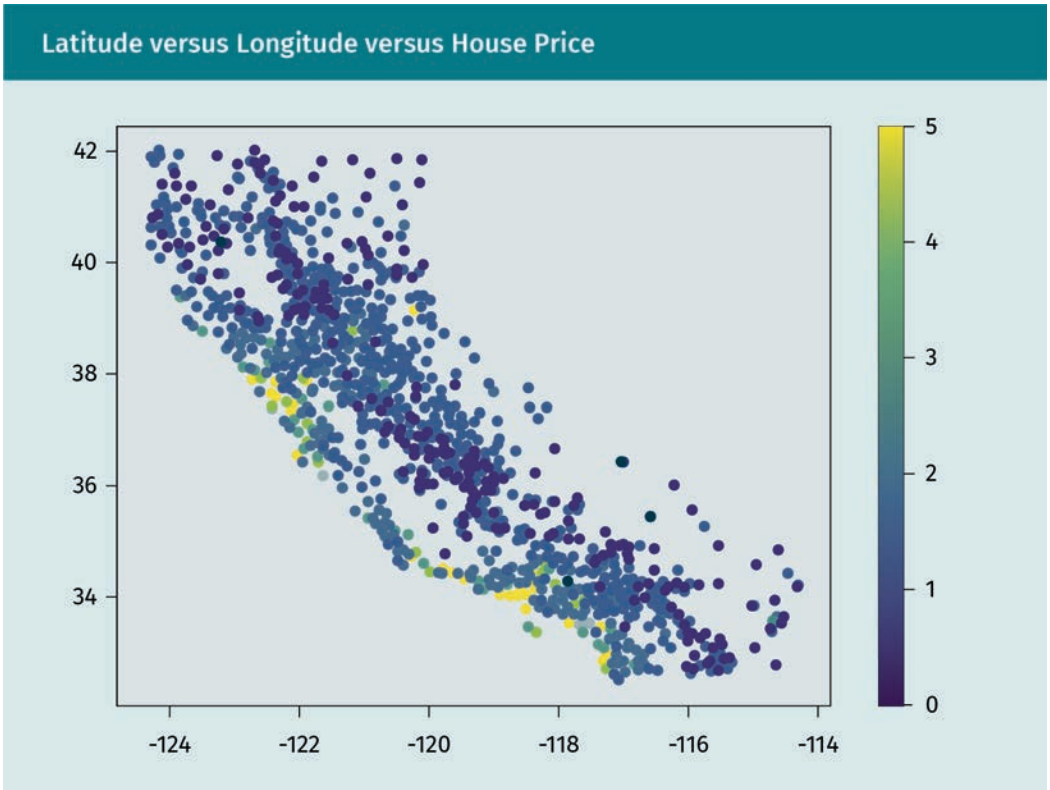
- Feature engineering. If we want to use various feature engineering approaches or exploit our domain knowledge, we should at least formulate a plan if this can only be done reasonably well during the modeling process. Ideally, we prepare as many steps beforehand as possible.
- Feature selection. Similarly, we should decide how we are going to select the most relevant features for our model, possibly controlling the false discovery rate (FDR) if we have many features. Univariate checks and the study of correlations can be done before the modeling phase begins as well.

As the above list indicates, building a predictive model is one of the last steps in the process of analyzing data or making data-driven decisions. Although it might be tempting to "throw" all data at some machine learning algorithm, a deeper understanding helps to build a better, less complex, and faster model. In particular, the role of data quality cannot be emphasized enough since the model is tuned using the available data, and any bias in the data can potentially be present in the final model. Many examples where model building has "gone wrong" can be found in Wachter-Boettcher (2017).

Hierarchical Models

In the simplest case we use all available feature variables in a single model. However, in many cases it may be more beneficial to split the model into several tasks. This has the advantage that each of the smaller models can focus on a specific sub-task or exploit the information of a subset of variables optimally.

Consider the California Housing dataset (Pace & Barry, 1997), which contains latitude and longitude as variables of the properties in question. The target of the model is to predict the price of the houses (on an adapted scale between zero and five). If we visualize the price as a function of latitude and longitude, shown below, we can see that prices are highest in densely populated areas, such as San Francisco or Los Angeles. Instead of using the raw variables latitude and longitude directly, we can build an intermediate model, for example, based on a clustering approach, to build a first estimate of the property prices.



This estimator is then used as a new feature variable in the final model. Hence, the final model depends on an intermediate model. Depending on how many of these intermediate steps are used, a complex hierarchy of models can be established. However, it should be noted that each additional hierarchical level increases the complexity as not only the final model needs to be trained but all intermediate models. Any change in the hierarchy requires the retraining of all steps involved. While this hierarchical approach can help exploit available data to its fullest extent, we should carefully consider if the trade-off between an improvement in the model performance is worth the increase in complexity.

Ensemble Models

Weak learner
A weak learner is a model or prediction algorithm which by itself is not expected to perform very well. The main advantages of weak learners are that they are often simple to use and train.

Many machine learning algorithms or modeling approaches, such as single decision trees, can be considered **weak learners** (Zhou, 2012; Schapire, 1990). These weak learners, as opposed to strong learners, are not able to exploit the data fully and build an optimal model. However, they might be conceptually simple or easy to use which makes them appealing for model-building approaches. The idea behind ensemble methods is to combine several weak learners into a group or ensemble with the aim that this group is more powerful than each individual member, i.e., several weak learners are combined into a strong learner.

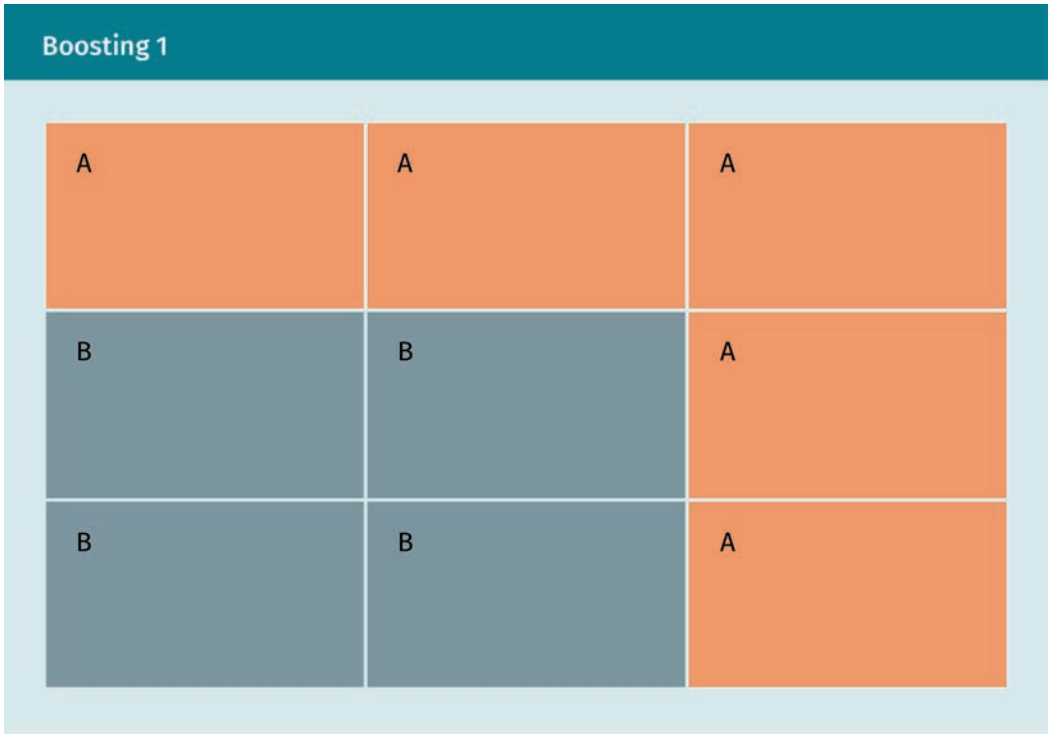
As an example, we can combine several decision trees into a random forest (Ho, 1995). The output of the model is then not the prediction made by a single decision tree but the combined output of the ensemble method. There are several ways in which the

final output can be constructed. For classification tasks, for example, we can use a majority vote where the final class decision is given by the class that has been identified most often by the individual trees in the ensemble. In case of regression tasks, we can take the mean value of all elements in the ensemble.

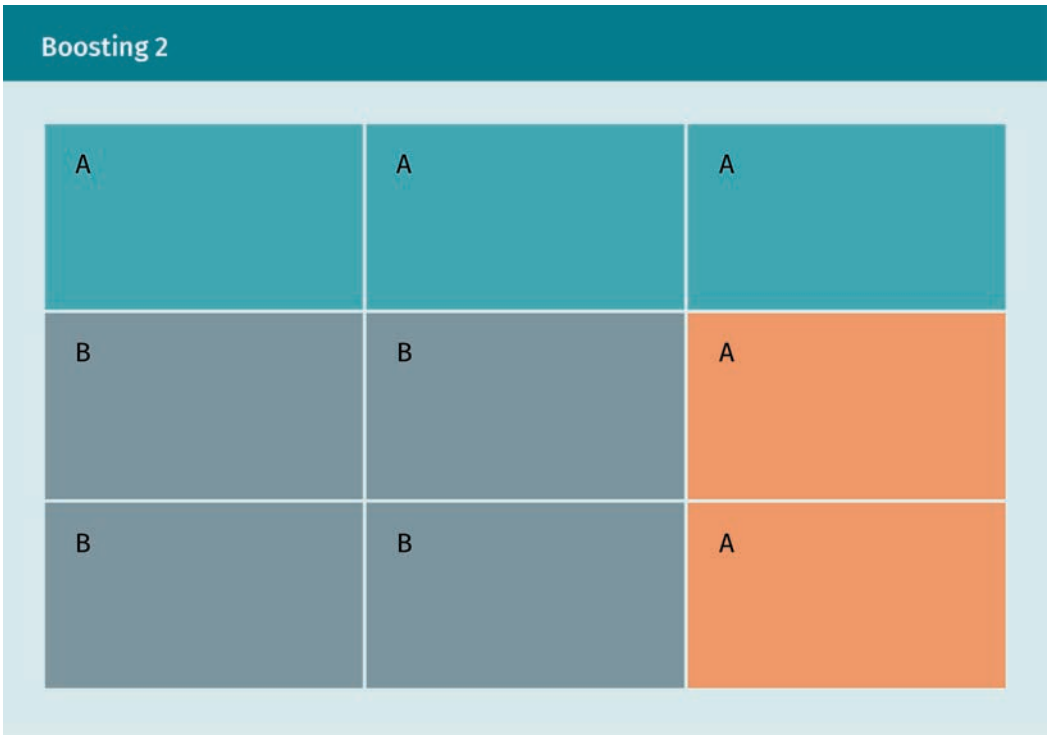
Boosting

Boosting (Schapire, 1990; Freund, 1995) is another way of combining several weak learners into one strong one. Instead of using a group of weak learners as an ensemble, the idea behind boosting is to build a sequential model where each step aims to correct the mistakes of the previous ones. This can be illustrated in the following way.

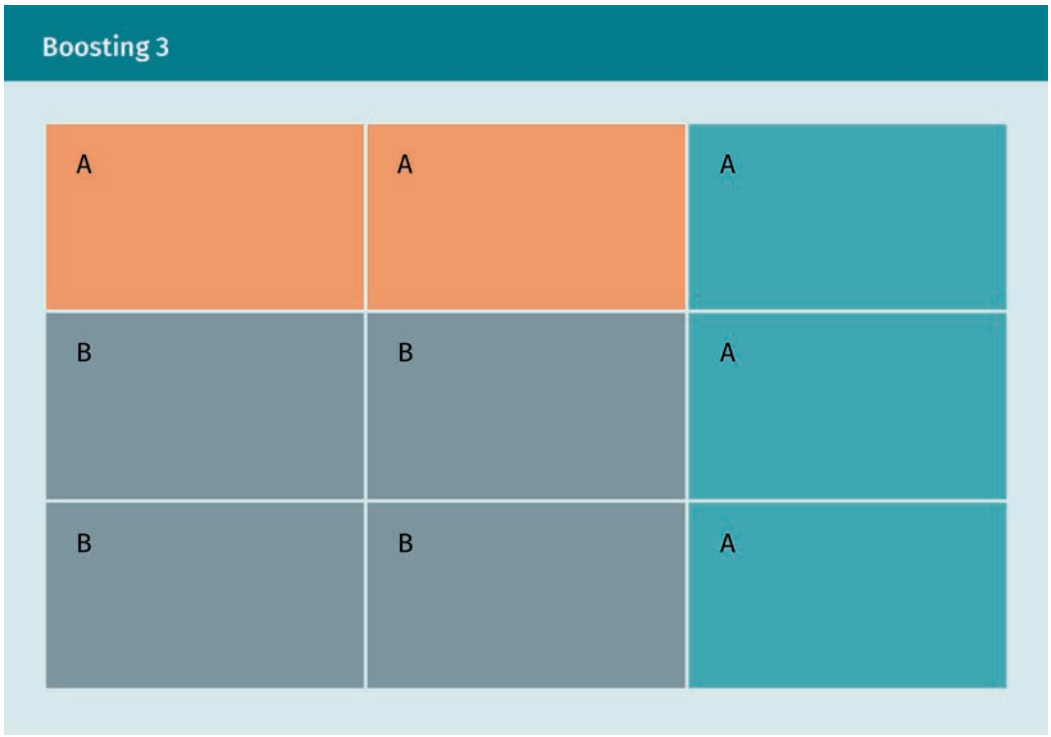
Suppose a model is built to separate class A from class B with the following setup:



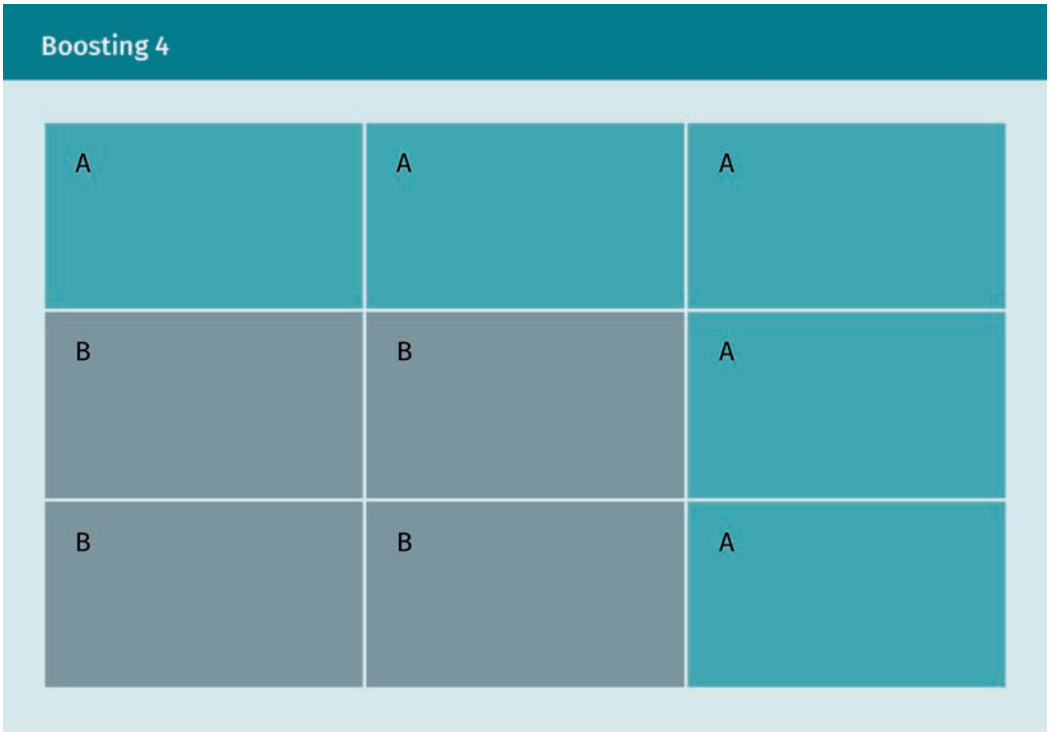
The first weak learner may have learned to separate the first row from the rest, as shown below.



In this case, most of the instances of class A (three out of five) are identified correctly; but, because the model is built with a weak learner, we can't expect it to get all predictions correct. We now add a second weak learner that tries to correct the mistakes the first estimator made. This can be done by, for example, assigning a higher weight to the instances of the class that weren't identified correctly. In the example above, we would increase the weights on the two missing instances of class A on the right. The second weak learner might give the following result.



The right column is now identified correctly but two of the instances of class A are now incorrect. Hence, we repeat the process and assign higher weights to these cases. Repeating the process iteratively, we finally arrive at a chain of weak estimators that predict all cases correctly.



This intuitive approach can be summarized as

$$F_T(x) = \sum_{i=1}^{T} f_i(x)$$

where $F_T(x)$ is the combined boosted strong learner and $f_i(x)$ is a single weak learner such as a single decision tree.

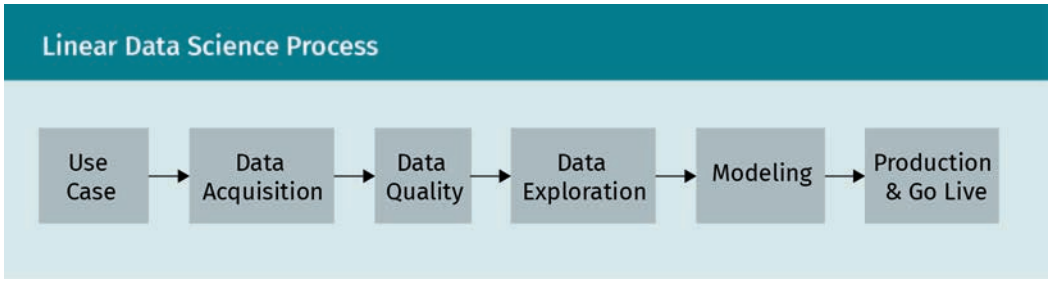
Implementations of this approach are, for example, AdaBoost (Freund & Schapire, 1997) or Gradient Boosting (Breiman, 1998; Friedman, 2001, 2002). The idea behind gradient boosting is that each new added weak learner corrects the output of the chain so far and minimizes the residual $R(x) = F(x) - y$, where $F(x)$ is the output of the current ensemble and y is the desired output of the target variable. This is equivalent to minimizing a cost function $C(x,y) = (F(x)-y)^2$, which can be done numerically using **gradient descent**.

Gradient descent
This is an optimization procedure where the parameters of a model are varied to determine the steepest descent (or gradient) of a specific cost function.

5.1 Establishing a Benchmark Model

In its simplest form, the data science workflow can be visualized as a linear chain of steps, as shown below. In most projects, the various steps are linked together; hence, both the process description CRISP-DM (Shearer, 2000) as well as Team Data Science Process (Microsoft 2017) add these dependencies with interconnecting arrows. However,

in each case the "modeling" step is interpreted as one entity, whereas feature engineering, model training, and model evaluation may be discussed as separate sub-entities.



Introducing a benchmark model isolates the various steps in the process. We can either use an existing model or procedure as benchmark model or create a new model. In case we create a new model, it is important to remember that the benchmark model is not intended as the final model while its predictions should be viable, but the model doesn't need to perform extremely well. It is more important that we can establish such a benchmark model relatively quickly (e.g., a few days at most), for example, using linear or logistic regression. Establishing such a benchmark model quickly has the following advantages.

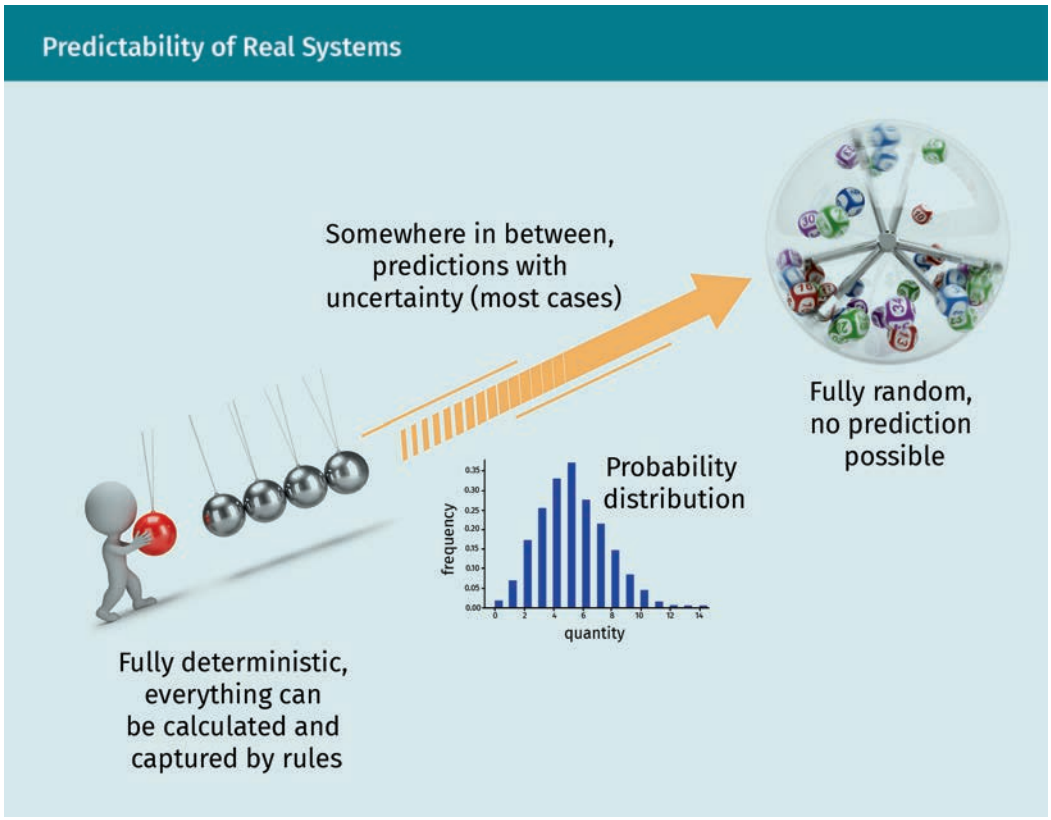
First of all, creating a benchmark model establishes a baseline. Irrespective of how our final model will be built, it should never perform worse than either an existing approach or a "quick & dirty" model. This baseline also helps judge how much the current state of the "real" model improves on the baseline. Initially, we expect that our modeling efforts will improve the performance of the model significantly compared to the baseline. However, at some point we will notice that the performance can only be improved by a small margin by adding more resources or spending more and more time. While this may be acceptable or even desirable in academic research or competitions, in industrial applications the work of marginally increasing the performance of a model has to be seen in the light of associated costs, both in terms of time spent on the improvement and additional resources we might need to acquire, clean, and process further data sources. In these cases, the benchmark model helps determine when a model is "good enough," i.e., when the further marginal increase in performance would not lead to a better fulfillment of the use case definition given the costs associated with this effort.

Next, the benchmark model allows us to determine whether the effort of building a more sophisticated and complex model is worth it from a maintenance and support point of view. In many cases, the model will need to be integrated into a production level pipeline, potentially operated 24/7/365. The more complex the model is, the more resources are likely required to operate it (both in terms of computing power and expertise to maintain the model). From a business perspective, we will have to decide if and how long the investment into developing a new model should be made.

Finally, establishing a benchmark model early on in the process allows us to separate the implementation of the technical pipeline from the work on the model itself. Since the benchmark model is indeed a valid model and produces predictions in the same way the final model will (albeit with a lower performance), the benchmark model can be used to implement and test all technical issues regarding data acquisition, processing, handing the data to the model, retrieving its predictions, as well as storage and persistency aspects. This allows the data engineering, operations, and support teams to build the entire infrastructure stack in parallel to the work on the model itself, significantly reducing the overall delivery time for the project.

5.2 Prediction as Probabilities

Realistic systems can be (broadly) categorized by how well we can predict their future behavior. There are few deterministic systems such as a Newton's cradle or grandfather clock. In fact, most of these deterministic systems are only deterministic if we don't look too closely. As soon as we take friction or heat dissipation into account, these systems also become more complex. Fully random systems such as the draw of lottery numbers are at the other end. In these cases, we can't predict anything. In either extreme, we don't need any sophisticated modeling approaches or machine learning algorithms: either the system can be fully described by a (complex) set of equations or its future behavior cannot be predicted. Nevertheless, most realistic systems lie somewhere in the middle.



This reality implies that there is some predictable, deterministic, or regular component which we can predict, as well as a random component that essentially increases the uncertainty or volatility of the prediction. Consequently, anything we wish to predict with a model or machine learning approach has to be treated in terms of probability.

In case of classifications, the prediction is not made as “class A” or “class B” but rather as a continuous probability between zero and 100 percent, and a suitable cutoff has to be defined that allows us to set a threshold when we attribute an event to either class A or B.

The exact placement of such a threshold depends on the use case and the application of the model. For example, if we wish to identify fraudulent credit card transactions, we could set the threshold to 95 percent probability that the transaction is fraudulent. This would identify truly fraudulent cases with high accuracy. However, depending on the nature of the transactions, we might miss many transactions that are fraudulent but don’t score as high. Alternatively, we might set the threshold much lower, e.g., 50 percent. In this case we would capture many fraudulent cases, but also risk a high proportion of false alarms that would have to be checked manually to avoid customer complaints. The preferable approach depends on the use case and business model where the predictive model is to be used.

Similarly, in the case of the prediction of a quantity, i.e., a regression model, the predicted quantity is not just a number but rather a probability density function. In the simplest case, we can predict a number for the quantity we need to predict as well as a

measure of its uncertainty. Implicitly, we then assume that the process follows a Gaussian distribution. The performance of the overall project can often be improved by a wide margin by not making this assumption but working with the correct underlying probability distribution. Depending on the use case and our domain knowledge, we may know which distribution to use. For example, sales of goods at retailers are typically described by a Gamma-Poisson or negative binomial model, the failure of components, or material fatigue by a Weibull distribution. In these cases we can use our model or machine learning approach to estimate the parameters of the model directly so that we can determine the full probability density function (PDF) for each individual prediction. If we don't know the probability distribution, we can model or learn the shape of the distribution empirically from the available data. In either case, we need to transform the predicted probability distribution for each event into a single number called the optimal point estimator. Using a suitable **cost function**, we can determine the quantile of the predicted PDF that minimizes the cost function

Cost function
In this context, the cost function details all the contributions to a function that defines the details of the effect when a prediction is lower than the true value as well as higher. For example, the MAD implies that the cost associated with missing the true value rises linearly both for over- or underestimation whereas the MSE implies a quadratic dependency.

$$\frac{\partial \mathrm{E}[\mathrm{C}[\mathrm{p},\mathrm{t}]]}{\partial \mathrm{p}} = 0 \text{ and } \frac{\partial^2 \mathrm{E}[\mathrm{C}[\mathrm{p},\mathrm{t}]]}{\partial \mathrm{p}^2} > 0$$

where p is the optimal point estimator and t is the true value in the training dataset. In the case of a linear cost function (MAD), the associated optimal point estimator is the median of the predicted distribution; in the case of a quadratic cost function, such as the MSE, the mean is the optimal point estimator. As in the above case for the classification, the choice of the optimal point estimator depends on the use case and the intended use of the model and its predictions. For example, a retailer may choose to use a 90 percent or even 95 percent quantile of the predicted distribution to avoid out-of-stock situations where the customers face empty shelves which increases the waste of perishable goods. Choosing the best compromise depends on the business strategy of the retailer and the priorities set in the definition of the use case. The field of use case delves deeper in this work.

5.3 Interpretable Machine Learning and Results

Machine learning approaches help build accurate predictive models for a wide range of applications. However, the more complex the models become, the more opaque and difficult they are to interpret. For example, deep learning models based on neural networks are often very powerful, but the path that leads to individual predictions and decisions is typically not obvious or easily understandable. These are referred to as **black box models**.

Black box models
These are models where we don't understand how an individual decision was made. We may understand the underlying theory

On the other hand, understandable decisions are sometimes required due to legal and regulatory constraints or are (highly) desirable in other cases. As an example, some (future) use case in predictive personal medicine may center on determining the treatment plan for individual patients, including the choice and dose of medicine recom-

mended for a given patient by a machine learning model. Even without a legal obligation, a detailed explanation could be found for both the treatment plan and the dose of the medicines, instilling and increasing patient trust.

but cannot follow the path all the way from the input to the output to understand why a specific output was calculated.

Fundamentally, we can follow one of two approaches in building interpretable models: we can either build models that are intrinsically interpretable or apply methods to black box models which aim at explaining each prediction a model has made in a separate step afterwards. A good overview about common approaches can be found in Molnar (2019).

Linear Models

The simplest fully explainable model is linear regression where the target variable is expressed as a linear combination of the features: $y = \sum_i a_i x_i + c$. The strength of the coefficients a_i indicates the importance of a given feature. Unfortunately, such a simple model suffers from many shortcomings and is rarely suitable for practical applications. For example, all features are assumed to be linear with constant variance as well as independent from each other. These conditions are rarely fulfilled in practical applications.

Generalized linear models (GLM) (Nelder & Wedderburn, 1972) extend this approach. The model consists of a linear predictor $\eta_i = \sum_i a_i x_i + c$ and a link function describing how the target variable Y depends on the mean $E(Y_i) = \mu_i$: $g(\mu_i) = \eta_i$. The link function relates the linear predictor to the mean of the distribution function. Generalized Additive Models (GAM) (Hastie & Tibshirani, 1990) expand on this concept and allow some function $f_i(x_i)$ that operates on the features x_j to allow for nonlinear effects. In general, GAMs are described by $g(E[y]) = \sum_i f_i(x_i) + c$. If the function $f_i(x_i)$ is constrained to be linear, GLM are recovered. GAM are not as easily interpretable as GLM due to the (nonlinear) function $f_i(x_i)$; however, they do retain most of the benefits and allow us to model more complex relationships. A recent example of GAM is Interpret by Microsoft (Caruana et al., 2015; Nori et al., 2019; Lou et al., 2013) which also allows for pairwise interaction of features, i.e., $g(E[y]) = \sum_i f_i(x_i) + \sum_{i \neq j} f_{ij}(x_i, x_j)$. In order to retain some interpretability, the pairwise interactions can be represented by a heat map.

Explainable Machine Learning

Looking at various machine learning algorithms, single decision trees (Breiman et al., 1984) are fully explainable: the tree is built by splitting the feature variables. However, similar to linear regression models, a single decision tree is often not suitable for practical applications. Again, all features are assumed to be linear and independent from each other for the split to work well. Extending this method by combining multiple decision trees into random forests (Breiman, 2001) increases the performance and makes it viable for practical applications; however, the models become less and less explainable. The same consideration holds for Support Vector Machines (SVM) (Boser et al., 1992). For a linear kernel, the weights define the hyperplane separating two classes. If there are only a few features, this can be used to understand how each decision was

constructed. However, in case of many features or nonlinear kernels, this is no longer possible. Artificial neural networks generally don't allow any insights into how a single decision was made and are hence considered black box models.

In the case of models that are not intrinsically explainable, we can follow several approaches to understand how individual predictions and decisions were made.

Surrogate model
The surrogate model has desirable traits (e.g., it can be used to explain a single model output) but is not as performant as the full model. We create a new model that is used in lieu of the original model for a specific model output.

Surrogate models such as LIME (Ribeiro et al., 2016) create a linear model for each prediction of a black box model that is locally faithful. This means that surrogate models such as LIME can be used to gain insights into each individual prediction and explain what contributed to this decision. However, the original model itself remains a black box model; we cannot gain insights into how the model generally works. In many practical applications, however, we only need to understand how individual predictions were made.

Further approaches have been implemented in Google's What-If Tool (Google Research, 2018). For example, we can edit or remove features for any data point and observe how the model behaves or changes. In addition, we can identify and compare **counterfactual** data points for classification (Wachter et al., 2018). The tool identifies data points that are the most similar to a current data point but belong to the opposing class.

Counterfactuals
These events have not but could have taken place. They address the question: "What would have happened if [a specific event] had happened?"

Understanding Features

Other approaches work by aiming to better understand the features that are used in the black box model. The idea behind feature importance is to randomize or shuffle the values of a given feature variable and observe how much a given metric, such as the MAD or MSE, changes with respect to the baseline. This concept was first suggested in Breiman (2001) for random forests and later generalized by Fisher et al. (2019).

The contribution of individual features can also be estimated using Shapley values (Roth, 1988; Shapley, 1953). The concept originates in game theory and focuses on the contribution of individual players in a cooperative game. The original setting describes a group (coalition) of participants (players) who achieve a common goal together or attain some form of result or payout. Each participant has their own unique skill set, i.e., the participants are not equal. The Shapley values define an approach to attribute each player's contribution to the final result. This works by averaging all possible coalitions of the players. For example, imagine you had the participants A, B, C. If player A was on their own, their result might be, say, 10 (in an arbitrary unit). Now B joins and the result might be 15, rising to 17 when C joins. However, what would happen if C joined before B? If B and C have very similar skills, C cannot add much value if B is already in the group and vice versa. However, that doesn't mean that C is a less valuable member than B *per se*—the order in which they joined the group matters. The Shapley value is then obtained by averaging all possible combinations of participants. In the case of machine learning, the features act as the participants or players, and the model is the coalition or cooperative game. The Shapley value is then a measure of

how important an individual feature is in the overall ensemble of correlated features available. A comprehensive framework based on this idea is SHAP (Lundberg & Lee, 2017).

Interpretable Models for Images

In case of images, we can replace parts of the image with a gray box before applying the machine learning model. This process is called partial occlusion (Zeiler & Fergus, 2013) and helps investigate how the output of, for example, a deep convolutional neural network changes in response to the altered image. If the gray box is moved across the image systematically, we can identify the regions that have the most effect on the output of the image classification. Saliency maps (Itti et al., 1998; Simonyan et al,. 2013) illustrate specific features of the image. The main idea is to rank the pixels of the input image according to their influence on some scoring function, e.g., to which class an image is attributed. This can be used to build the saliency map, i.e., an image representing the pixels that need to change the least in order for the output to be affected the most. A convenient implementation for both partial occlusion and salience maps is Picasso (Henderson & Rothe, 2017). In case of (deep) convolutional neural networks used to analyze images, de-convolutions (Zeiler & Fergus, 2013; Zeiler et al., 2011) can be used to gain deeper insights into what this convolutional neural network “sees”: working on specific layers of the network, use the same components as a convolutional neural network, but work backwards and extrapolate the activity back to the pixel input of the image. This approach allows us, for example, to visualize the features a convolutional neural network has learned during the training process on a given stack of images in a training dataset. Alternatively, we can work backwards and analyze how a typical picture should look to be classified into a given category, for example, using Google's “DeepDream” approach (Mordvintsev et al., 2015b, 2015a).

Summary

Building predictive models is the core of model engineering. However, before completing this step, we need to understand the use case, know how the model will be evaluated, and make the necessary data available and clean.

The components we use to build a model depends on the task. Advanced approaches, such as ensemble methods or boosting, can help to improve the performance of the model we build; in some cases a hierarchical approach may be helpful. Establishing a benchmark model early on in the modeling phase helps set a baseline against which we can compare all further efforts and separate the technical implementation of the production pipeline from the work on the model itself.

A key part of building high quality models is remembering that, ultimately, everything has to be expressed in terms of probabilities, which have to be transformed to classes or numbers for operational purposes. This step has to be done with the concrete requirements of the use case in mind to be able to fulfill its objectives optimally.

In many cases, interpretable models are either legally required or highly desirable to understand how individual predictions and decisions were made. A wide range of possible approaches, from intrinsically explainable models to explanation of features or surrogate models, exist to address this issue.

Knowledge Check

Did you understand this unit?

You can check your understanding by completing the questions for this unit on the learning platform.

Good luck!

Unit 6

Avoiding Common Fallacies

STUDY GOALS

On completion of this unit, you will have learned ...

... which common fallacies exist.

... how to avoid overtraining and improve generalization.

... the difference between overtraining and overfitting.

... how to improve operational aspects by automating the workflow.

... how to persist and serve models.

6. Avoiding Common Fallacies

Introduction

When building predictive models there are many traps and pitfalls we need to avoid to build high quality models. These concern both the models themselves as well as the way we build them. Ideally, the models should be sufficiently complex to entail the essence of the data we use to build the model—but not more than this. Any additional complexity risks the models learning spurious relationships that are not a fundamental property of the system from which we obtain the data (e.g., due to statistical fluctuation). Overly complex models also risk learning all data by heart and are hence not able to generalize well to unknown future events. These risks can be approached using the various approaches discussed in this unit.

The other obstacles we might face are concerned with the way we work. In the beginning, we set out to build a "good model"; however, this quickly escalates into many approaches, tries, and avenues we have pursued. It's all too easy to lose track of all attempts we have made so far, what their results were, or how to revert back to a previous situation. Stringent bookkeeping and scientific methods help to address this issue. Various tools exist to make our life a bit easier.

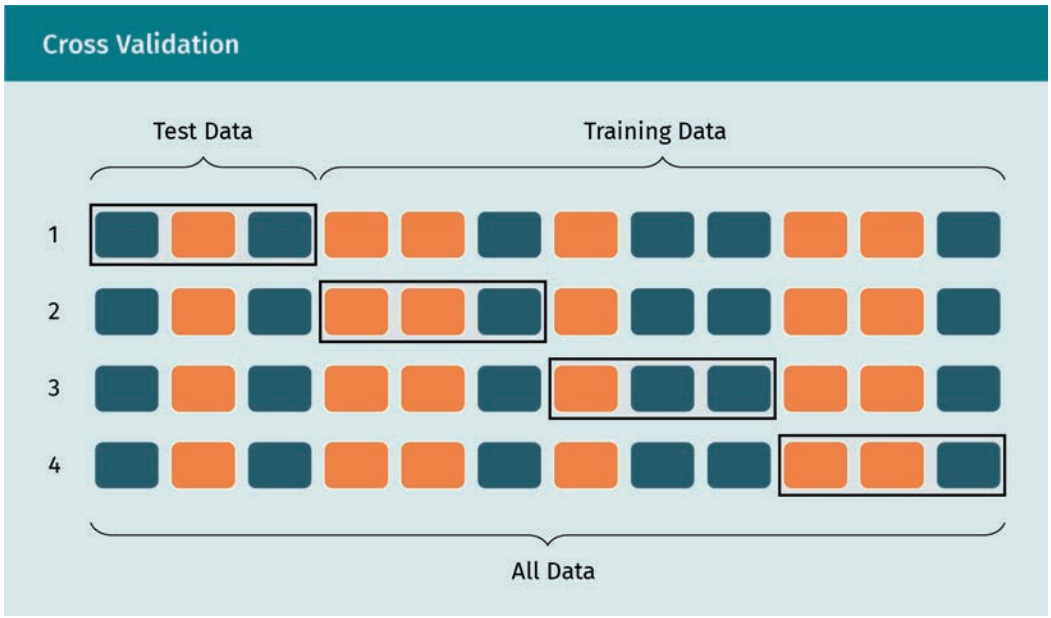
6.1 Overtraining and Generalization

The key objective of building a predictive model is to capture the essence of the data but not learn all the specific behavior of the data points used in the training by heart. In other words, we aim for a high generalization ability of our model where the model performs well on previously unseen or new data. Known as overtraining, the model can accurately describe the training data but fails to make suitable predictions for future or unknown events. A core part of building models is making sure to avoid this and retain high generalization abilities of the model.

Train/Test Sample and Cross Validation

In order to make sure that our model isn't prone to overtraining and retains high **generalization** abilities, we typically split our available dataset in two parts. The training data, which are used to build the model, and an independent test dataset, which is used to test the model. In most cases, we randomly assign about 80 percent of the data points to the training data and 20 percent to the test data. However, depending on the use case or peculiarities of the dataset, more details have to be considered. For example, if the dataset contains few rare events, we need to make sure that the training sample is sufficiently representative of the overall system while retaining enough events to test these cases. If our dataset contains imbalanced classes in classification, we also need to make sure that we address this both in the training and in the test data to avoid any bias.

In many cases we will not have the luxury of splitting the data into separate samples because we won't have enough data to cover all cases. This happens frequently even in the age of **big data** (Laney, 2001), especially when rare events need to be considered. In these cases, we can use cross-validation (Geisser, 1993; Kohavi, 1995; Devijver & Kittler, 1982). The idea behind cross-validation is shown below. Although the data are split into training and test data, as we repeat the process iteratively, different parts of the data are used as test and training data. This has the advantage that all data are used for training eventually; however, the model needs to be trained as often as we split iterate over the cross validation approach. This ensemble of models then needs to be combined appropriately, e.g., by using a majority vote for classification or the average for regression.



The following variants of cross-validation are often encountered:

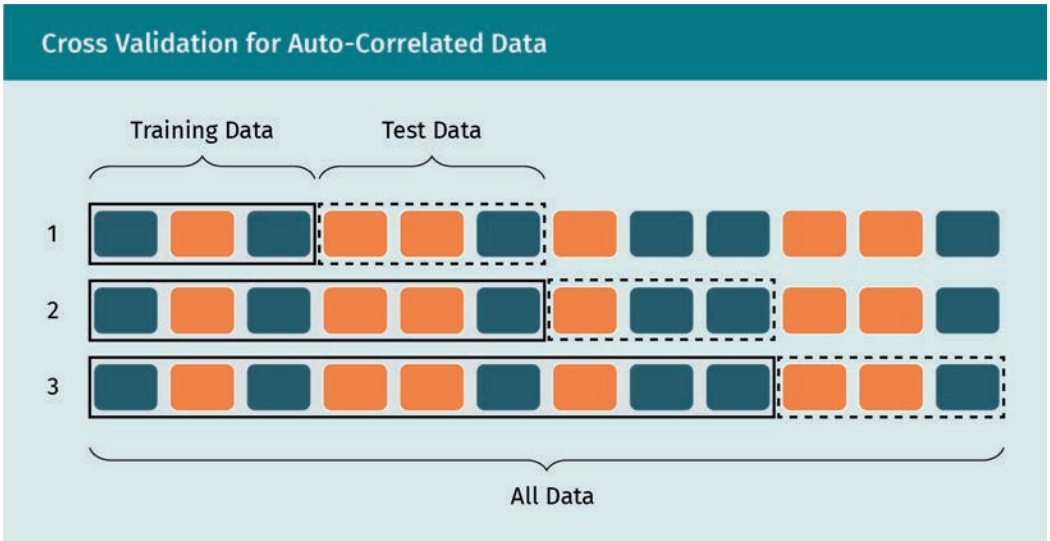
- K-fold cross-validation, where the data are split into k different samples, similar to the image above
- Stratified k-fold cross validation, where each sub-sample has the same fraction of the target variable as the full sample. If the classes are not equally populated, the stratified cross-validation retains this difference in the distribution of the classes.
- Leave-p-out cross validation, where only p events are removed from the full data sample
- Leave-p-label out cross validation, where p events are removed from the sample according to some external label. Instead of choosing the data points randomly, we use an external selection mechanism.

It is important to highlight that this cross-validation approach can only be used for independent data points, i.e., when one event described by a set of values of the feature variables is independent of the next and the order in which the data are taken does not matter. However, when we are using auto-correlated data this approach can-

Generalization
In the context of machine learning and modeling, generalization means that the model can predict future and unknown events well and that the model has not learned any spurious relationships or statistical fluctuations found in the training data.

Big data
This term is defined as "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" (Gartner, n.d.).

not be used out of the box. For time series data or for images showing a sequence of events, for example, each new data sample has a specific order in the overall sample. If the events were shuffled and randomly assigned to training and test data during the cross-validation process, this auto-correlation would be destroyed, preventing the model from learning the correct sequence. For example, if the data were taken in the month of January, February, through December, the values of the data will likely show a seasonal pattern. If we change the order of the data or take, e.g., the months March, April, May out of the data and use them as test data, the model cannot learn the correct sequence of events as we jump from February to June. In these cases, we need to train the model up to a point, use a specific window as training data and then the following window as test data. Then we extend the training window to include the test data from the previous round and use the next window as test data as shown below.



Hyperparameters

In machine learning approaches, hyperparameters are those parameters that are part of the overall model but not derived from the data during the training process of the machine learning algorithm. For example, as part of feature engineering we build early or simple estimators of our **target variable** to exploit some domain knowledge we might have and these new variables depend on some set of parameters. In contrast, the model parameters of the machine learning algorithm such as the weights of a (deep) neural network are not hyperparameters but model parameters.

Target variable
The target variable, also called the label or dependent variable, is the variable that should be modeled or predicted.

These hyperparameters also depend on the data and need to be set to a specific value before the machine learning model is trained. This implies that the machine learning model implicitly depends on the choice of the values of the hyperparameters. If we don't know which value these hyperparameters should take from external or domain knowledge, they need to be estimated from data as well. Which data should be used for this? If we use the training data, we use the same dataset twice, first to tune the hyperparameters and then to train the machine learning model. Hence, the two steps

are not independent as they are performed on the same data, risking the introduction of a bias. If we use the independent test data, we risk accidentally leaking some information about the unknown data (with which we want to evaluate our model performance) into the training data through the values of the hyperparameters. The test data would no longer be independent from the training data and, hence, this approach should not be followed. In an ideal case, we can split out data into three independent samples: one for optimizing the hyperparameters, one for training the machine learning algorithm, and one as an independent test sample. However, even in the age of big data, this is often an unaffordable luxury, especially if the dataset contains rare events that need to be included in the modeling process.

Several approaches exist to determine the value of the hyperparameters. In a "brute force" approach we can define a grid of values of the hyperparameters to test, i.e., for each hyperparameter we define a range and step size in which the values should be tested. However, this approach is difficult to implement if we need to determine the value of more than a few hyperparameters as each value has to be tested against each combination of all other hyperparameters. Alternatively, if we know (or can define) a probability distribution that defines the likelihood for each value of a hyperparameter, we can then randomly sample these distributions and pick a value of the hyperparameters from them. This has the advantage that the time we invest in the determination of the best values of the hyperparameters is independent from the number and range of the parameters. The more time we can invest, the better we can explore the **phase space** of the possible values. Several tools, such as Optuna (Akiba et al., 2019), exist to help optimize the hyperparameters of objective function.

Phase space
The phase space is defined by all possible values of all hyperparameters. Each point in this space corresponds to a specific combination of numeric values.

Beyond the hyperparameters discussed above, there is also a second set of hyperparameters used in machine learning algorithms. These are the values which are used to configure the algorithm. Hyperparameters in (deep) neural networks include: the learning rate, number of **training epochs**, and the strength of any regularization constant.

Unlike the other type of hyperparameters, these parameters are typically not determined from data beforehand but either set to specific values manually or, ideally, adapted dynamically during training, reducing the need for manual intervention.

Training epoch
A training epoch is one full iteration of the training process of a machine learning model where all training events are considered at least once.

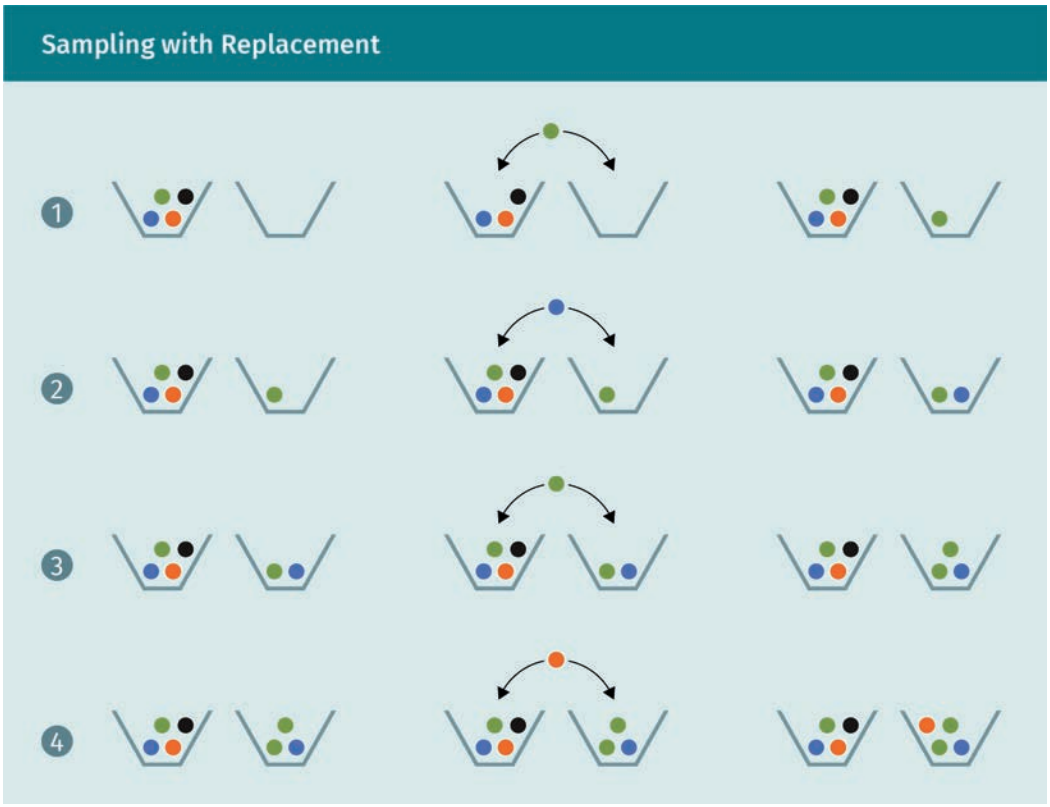
Bagging

In many cases, we will use weak learners such as decision trees as the starting point of our modeling effort. However, weak learners have a high risk of overtraining and learning the noise present in the data. Bagging (Breiman, 1996) addresses this by combining many trees into a forest, where each tree is trained on a subset of the data. Hence, no single tree sees all the data, only their combination.

The creation of a new dataset based on the original is illustrated below. We aim to create a new dataset from the original that has as many entries as the original one that is not a 1:1 copy. If the original dataset has N entries, we repeat the following approach N times:

- Take the “bag” of the original and derived dataset.
- Choose an element of the original dataset at random.
- Place a copy of the element into the new bag and return the original item into the original bag.

At the end of the procedure we now have two “bags” with the same number of entries but a different composition. Note that this implies that some of the data points will occur multiple times in the new “bag.”



Each weak learner is now trained on one of the new “bags” of data and the ensemble of these weak learners is combined into a single strong learner, e.g., majority vote for classification or averaging the regression.

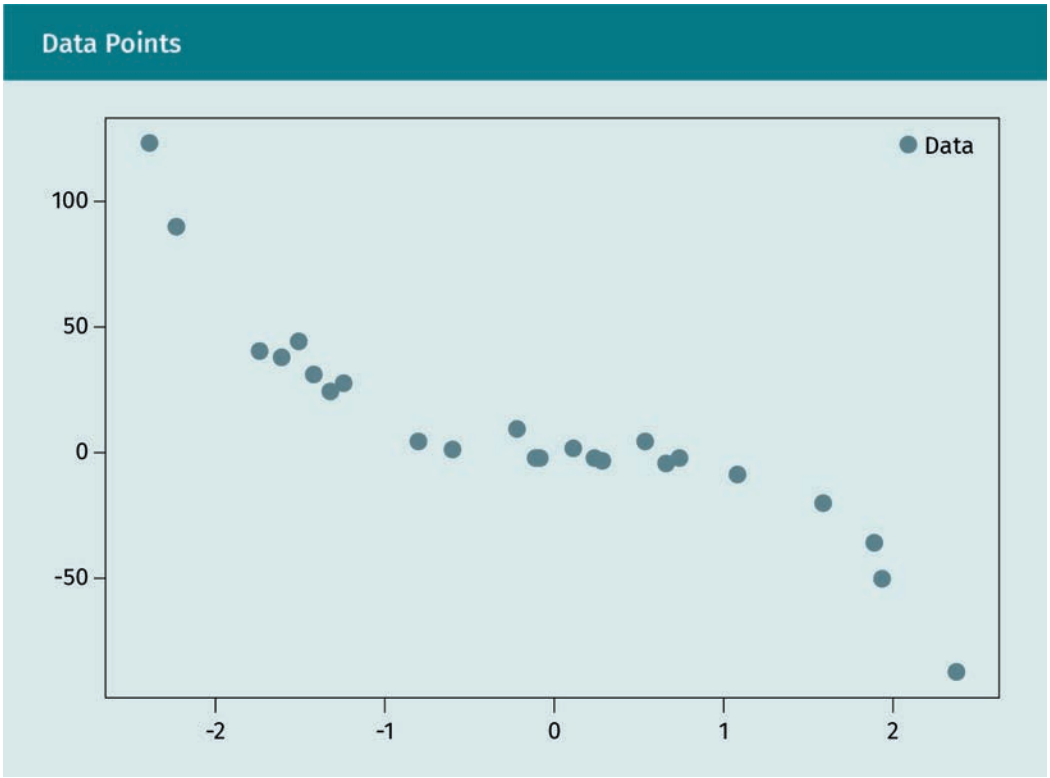
Feature bagging (Bryll et al., 2003; Ho, 1998) follows the same idea but instead of creating N different “bags” of data points, different “bags” of features are used. Each weak learner uses a different combination of feature variables.

Both bagging approaches can of course be combined, if both methods are applied to decision trees the resulting ensemble is called “random forests” (Ho, 1995).

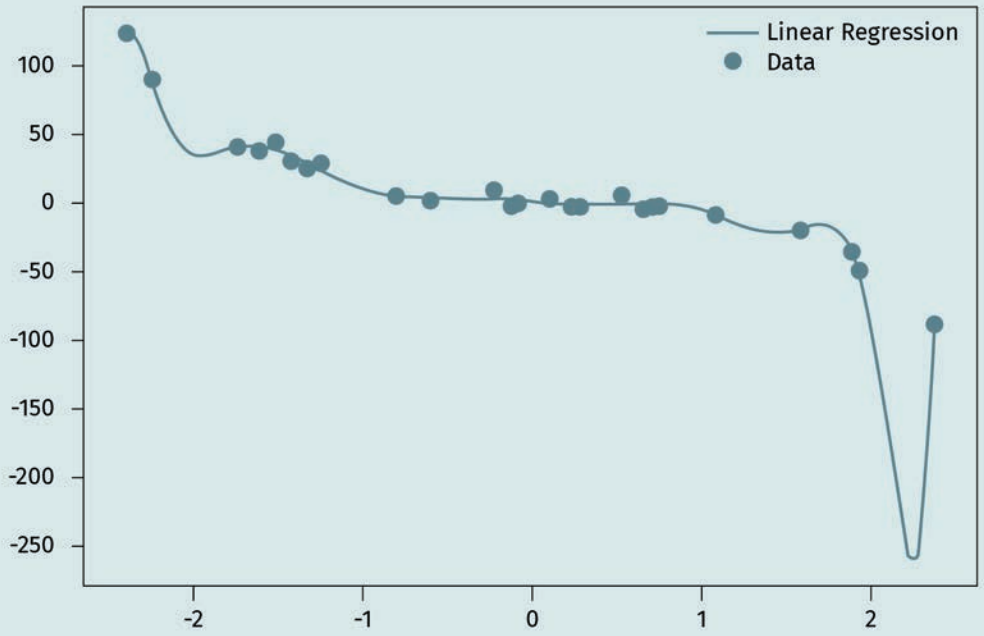
Regularization

The general idea behind regularization is to add constraints to the objective function of a model or machine learning algorithm so that the model cannot vary "too wildly" during the training process.

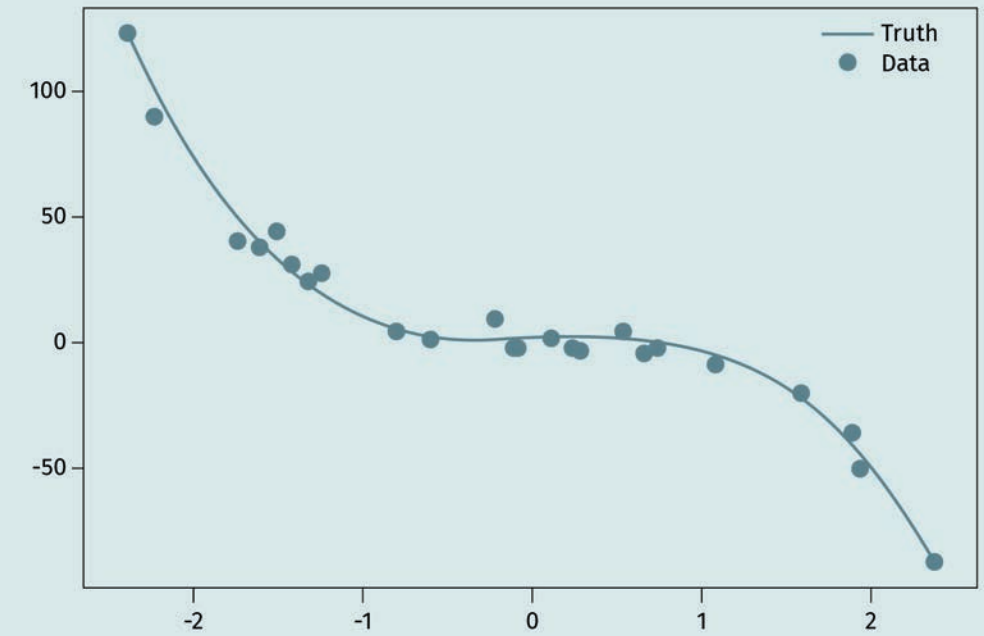
This can be illustrated using linear regression where the target variable is estimated by a linear predictor: $\hat{y} = \sum_j a_j x_j + c$. Hence for each data point y_i we look for the best predictor $\hat{y}_i$ such that, e.g., the difference $\sum_i (y_i - \hat{y}_i)^2$ is minimal by adjusting the free set of parameters a_j. The first graph below shows some fictional data points which we want to fit using linear regression. If we just apply the above formulae we obtain, e.g., the result shown in the linear regression below. Our objective is fulfilled—each data point is described exactly by our regression model. However, intuitively we would expect that it has too many degrees of freedom and "wiggles" too much. In this example we do have access to the function from which the data points were generated. Looking at the last graph, we see that the function does not show any "wiggles"—as we would expect intuitively. Hence, we need to change our linear regression model to suppress these "wiggles."

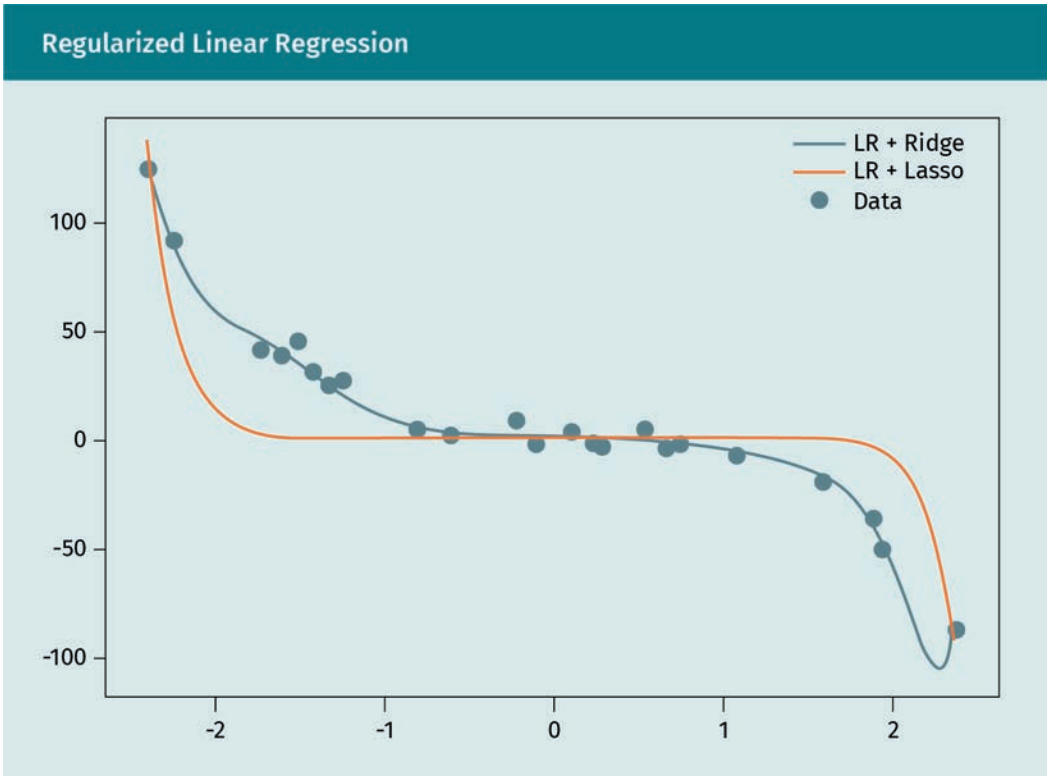


Linear Regression to the Data Points



Generating Function for the Data Points (Truth)





Adding such a suppression term is the idea behind regularization. We add a penalty to the function we want to minimize. In the case of the linear regression, we now minimize the model $\sum_{i=1}^{N}\left(y_i - \sum_{j=1}^{p} a_j x_j\right)^2 + \text{penalty}(a)$ where the index i indicates all data points and j denotes our feature variables. The penalty acts on our free parameters and adds additional constraints. We now need to decide what kind of penalty to add and how strong the penalty should be. In (linear) regression, the following approaches are popular:

- Ridge regularization (Tikhonov, 1943, 1963; Tikhonov & Arsenin, 1977; Tikhonov et al., 1995; Tikhonov et al., 1998)

$$\text{penalty}(a) = \lambda \sum_{j=1}^{p} a_j^2$$

- Lasso regularization (Santosa & Symes, 1986; Tibshirani, 1996):

$$\text{penalty}(a) = \lambda \sum_{j=1}^{p} |a_j|$$

where the parameter λ determines the strength of the regularization. This parameter has to be determined as well and is a hyperparameter of our model. Adding such a regularization scheme to our linear regression model results in a "smoother" curve as shown above.

Individual decision trees are prone to overtraining. For any given data we can always construct a tree that predicts the outcome perfectly by adding an arbitrary large number of branches to the tree, making it very deep. By construction, the leaves are "pure" if they only contain one data point, i.e., if the trees are optimized during training by optimizing the purity of the data in the leaves, we have to prevent that leaves are created with only one data point in them. The simplest approaches to avoid overtraining in decision trees are therefore to require a minimal number of data points associated with the leaves as well as limiting the number of branches that can be created during the training process. Random forest (Ho, 1995) also uses bagging and feature bagging for the individual trees in the forest to reduce overtraining as discussed earlier.

In addition, we can "prune" decision trees to further reduce the risk of overtraining. This approach is inspired from the way a gardener would cut (or prune) branches of real trees to shape their growth. In machine learning, the general idea is to start from a full-grown tree and then prune those branches away that give rise to overtraining by starting from the leaves and work toward the root of the decision tree. The intuition behind this is that once we have started to grow the decision tree we may find that not all splits (nodes) and the subsequent sub-trees are optimal. Had we known the future splits towards the later branches, we might have chosen a different split earlier in the process. In weakest link pruning we start from a fully grown tree T_0. We then replace a sub-tree with a leaf and obtain a new tree T_1. As we do this iteratively, we select the sub-tree that has the least impact on the metric we use to evaluate the performance of the tree. For example, if we use the residual sum of squares for a single variable, we can determine how much the performance of the new tree T_1 is degraded from the original one with

$$\frac{\mathrm{RSS}(\mathrm{T}_1) - \mathrm{RSS}(\mathrm{T}_0)}{|\mathrm{T}_0| - |\mathrm{T}_1|}$$

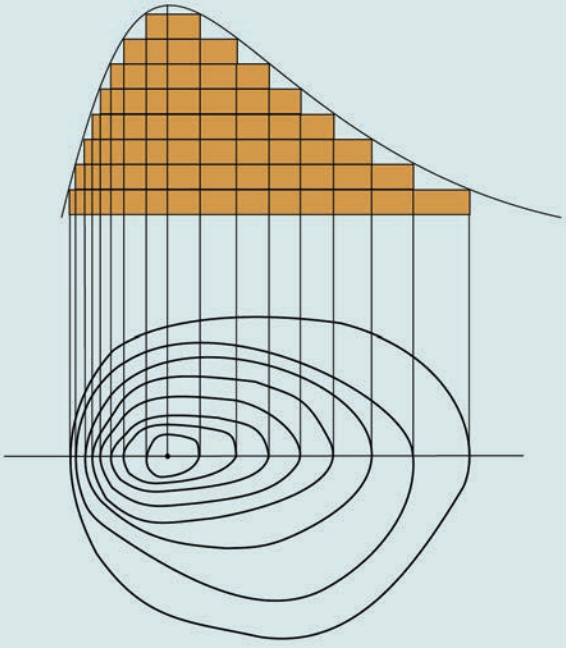
where we normalize to the difference of the number of leaves of the original tree ($|\mathrm{T}_0|$) and the new tree ($|\mathrm{T}_1|$). Note that the residual sum of squares (RSS) is defined as $\mathrm{RSS} = \sum_\mathrm{i} (\mathrm{y_i} - \hat{\mathrm{y}}(\mathrm{x_i}))^2$. The process is repeated iteratively until we replace the entire tree with a single leaf again and we can use cross-validation to choose which pruned tree has the least impact on performance while reducing the complexity as much as possible. In the same way discussed earlier for Lasso and Ridge regularization, we can also modify the cost function which is used to optimize the decision tree during the learning phase. By adding a penalty we can minimize

$$\mathrm{RSS}' = \sum_{\mathrm{i}=1}^{\mathrm{N}} (\mathrm{y_i} - \hat{\mathrm{y}}(\mathrm{x_i}))^2 + \alpha|\mathrm{T}|$$

where $|T|$ again represents the number of leaves of the tree and α is a new free hyper-parameter that determines the strength of the regularization. For $\alpha \to 0$ the tree is again unregularized; $\alpha \to \infty$ forces the tree into a single leaf. The best value for α can be determined using cross-validation.

Neural networks are typically trained by gradient descent (Cauchy, 1847) using the back-propagation algorithm (Werbos, 1994; Rumelhart et al., 1986). The main idea is to iteratively change the free parameters of the neural network, the weights, such that a given cost function (like the residual sum of squares) is minimized. The cost function can be "visualized" as a high-dimensional hyperplane which is determined by all the values of the feature variables and the current prediction the network makes at each stage of the training. For two variables (or dimensions), this can be illustrated as in the figure below. Depending on the value of the variables, the cost function will have some value which can either be represented by a height, adding a third dimension to the figure, or by the densities of lines.

Illustration Gradient Descent

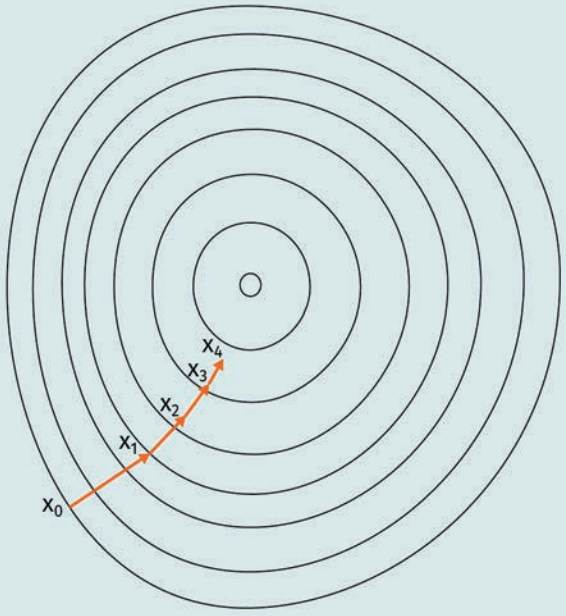


Romary (Wikipedia) CC BY-SA 3.0

In plain gradient descent, we would follow the line of steepest descent towards the next minimum (shown below) at each step of the optimization and training process. Although the surface of the gradient is generally not a smooth function, we can imagine a high-dimensional "mountain" with many cracks, valleys, cliffs, or ledges. As we move towards the minimum, we can easily lose sight of our target, as the next step going down the steepest path locally may well be in a different direction from the pre-

vious step as we face another cliff or an obstacle rising directly in front of us. The idea behind adding momentum is to maintain the general direction of travel. Irrespective of the direction we would choose at any particular step, we always retain the general direction of travel defined by the direction over the last few steps. This adds two more hyperparameters, namely how many steps to take into account when defining the previous direction of travel and the strength at which this direction should be kept. This way, small local perturbations of the complicated structure of the cost function we minimize have less of an impact. However, it becomes more difficult to change the optimization path: If we notice that the optimization is not on a suitable path and we need to change course, we can only do this slowly.

Illustration of the Optimization Path in Gradient Descent



Olegalexandrov, 2004. In the public domain.

A further approach to regularize neural network follows the same general idea as pruning in decision trees. Since neural networks have many parameters and have hence a high risk of overtraining, we aim to build the smallest possible network that allows us to solve the task. This can be done by removing the weights and connected network nodes during the training process which are deemed to be insignificant. This process is called Structural Risk Minimization (SRM). This approach can be implemented, for example, using weight decay described in the following, and all network weights that fall below a certain threshold are set to exactly zero, indicating that they are no longer part of the neural network. If all weights leading into a node are exactly zero, this node also no longer contributes. During weight decay, all network parameters (or weights) are forced to take the smallest possible values. This can be implemented by adding a

penalty term to the cost function based on the weights in the same way as discussed earlier for lasso and ridge regularization. For example, we can extend the cost-function of a neural network as

$$\mathrm{E_r} = \mathrm{E} + \lambda \|\mathrm{w}\|$$

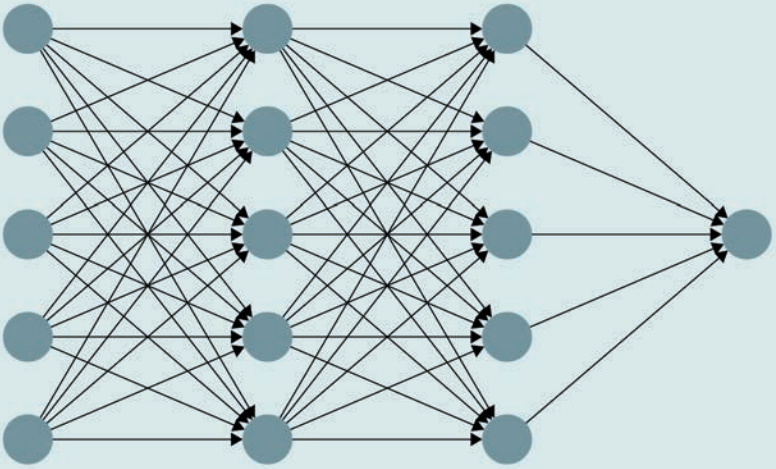
where E is the unregulated cost function (e.g., the sum of squares or cross entropy), λ indicates the strength of the regularization, and $||\mathrm{w}||$ defines a regularization metric (e.g., L_1 norm ($|\mathrm{w}|$) where we take the absolute value of the weights or the L_2 norm using the weights squared (w^2)). This added penalty "forces" the network to keep the relevant network weights "alive," especially when combined with SRM.

Finally, it can be shown that adding noise to the training set of a neural network is in some circumstances equivalent to adding a regularization term during the network training (Bishop, 1995).

As with many other approaches, we can combine several neural networks into a larger ensemble using bagging and feature bagging. We should keep in mind that neural networks typically have many free parameters that need to be determined during the training process and combining many neural network amplifies the challenges to determine all parameters well. This challenge is addressed by dropout (Srivastava et al., 2014). The main idea behind this regularization approach is to randomly "drop" network nodes and all their connections during network training effectively turning a neural network into an ensemble of smaller neural networks.

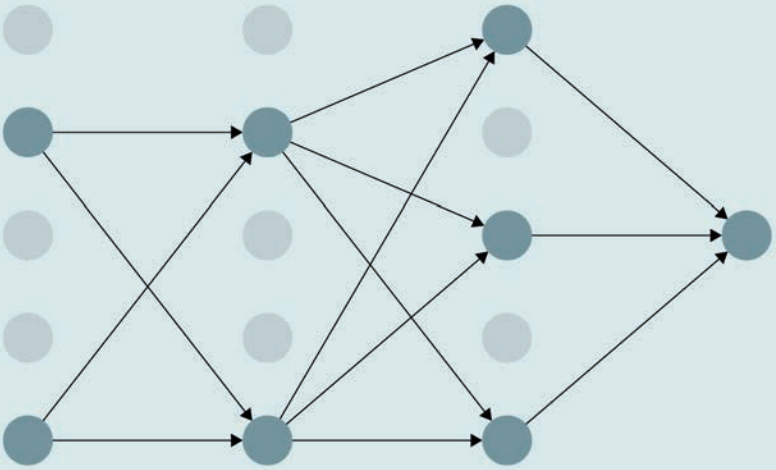
A typical neural network architecture is shown below where each network node is fully connected to all following nodes through the network weights. When dropout is applied, each network node is removed with some probability p, along with all the incoming and outgoing connections of that node. The resulting network will have significantly fewer connections as shown in the second image below.

Example Neural Network Architecture Prior to Applying Dropout



This process is repeated starting from the full network at each training iteration so that in each iteration a different set of nodes is removed along with their connections. Intuitively we can understand this as training many small networks with the additional constraint that all networks use the same shared weights, where each small network is rarely trained. As the nodes have been removed during the training process with probability p, this must be compensated when a prediction is made as at this time the full network is used. Hence, during the prediction time each node is always present, and its weights have to be multiplied by the probability p.

Neural Network Architecture after Applying Dropout



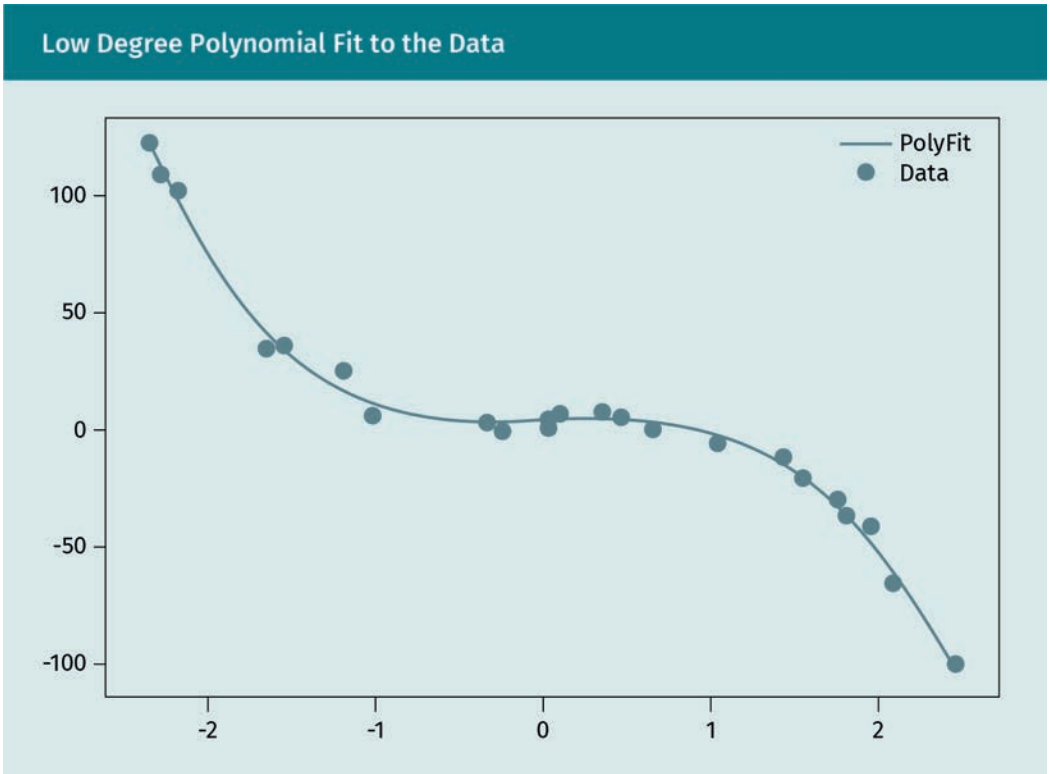
6.2 Overfitting and Occam's Razor

If overtraining and overfitting sound closely related it is because they are. In the previous section, we discussed overfitting which is mainly concerned with making sure that the model or machine learning approach only learns the relevant relationships from the data but not the statistical fluctuations. In an extreme case, a model would have enough parameters to learn the behavior of the training data by heart but would then struggle to predict future or unknown events.

Overfitting is concerned with controlling the complexity of a model. We previously encountered this when we discussed regularization. In the data points graph in the previous section, we started from a given dataset and used a very flexible model in the linear regression graph to describe the data. Although we intuitively recognized that this model describes the data well, it is not what we would have expected, and we introduced a penalty term to regularize our model.

In general, we can always describe our training data perfectly or minimize a cost or objective function by making our model or machine learning algorithm more complex. To avoid overfitting, we need to make our model as complex as it needs to be to describe the training data, but no more.

Going back to the linear regression graph, the polynomial model used to fit the data has too many degrees of freedom. While it is able to describe each element of the (training) dataset correctly, we don't expect it to capture the behavior of the data as well as the generating function graph, which shows the underlying true distribution that was used to generate the data sample. The distribution is quite smooth without "wiggles"; intuitively, we expect our final model to look similar. Hence, in addition to regularization, we need to control the complexity of the model. In this simple example, we can reduce the degree of the polynomial function used in the model to reduce the complexity, avoiding overfitting. As is shown below, instead of adding a regularization penalty, we have reduced the complexity of the model. We find that the model both describes the data well and behaves in the way we would intuitively expect.



More generally speaking, controlling the complexity is not limited to reducing or tuning the number of model parameters that need to be determined during the training process, but also includes critically reflecting on the choice of modeling approaches. As a guiding principle, the least complex model should be chosen to tackle the problem. The complexity should only be increased if a simpler model is no longer sufficient to perform a certain task. For example, if a linear regression model or a generalized linear or additive model is sufficient for a specific problem setting, it is generally preferable to use this instead of using a complex neural network-based approach in a "one-size-fits-all" attempt.

This principle is related to Occam's Razor, attributed to William of Occam, which (supposedly) states *pluralitas non est ponenda sine necessitate* (plurality should not be posited without necessity) (as cited in Duignan, 1998). The question of who said what is a bit lost in history (Thorburn, 1918).

Irrespective of the origins of the saying, it has served as a guiding principle for developing scientific ideas and models in all disciplines across the centuries. Models or explanations should not be more complicated than they need to be and it's generally best to choose the simplest possible model or explanation which explains an observation, dataset, or solves the problem. This does not only refer to data science but serves as a general approach, for example:

- A flat tire is due to a nail on the road rather than a gang of professional vandals specializing in tires.
- A child with a runny nose probably has a cold and not a genetic disease.
- A tree fell over in the forest because of a storm rather than woodworking aliens.

6.3 Workflow Automatization and Model Persistence

Data science projects vary greatly in project settings and expected outcomes. In some cases, we want to understand a specific aspect, and the project is focused on improving our knowledge about a specific application domain. In other cases, the project may be focused on feasibility studies or research and development (R&D), resulting in a prototype. Very often a data science project is pursued to create continuous **value** for a specific use case by developing and implementing a model in a continuous pipeline where data are processed at regular intervals, predictions about future events are made, and these turned into actionable decisions. The possible applications range from retailers who might want to optimize and automate their prices and supply chain, factory operators need to define the next step in the production pipeline, to banks needing to assess if a given credit card transaction is fraudulent.

Value
In a data science use case, value is used as a generic term that describes the impact of the results. In a commercial setting, this could be the return on investment, the savings due to some optimization, or the entrance into a new business domain.

As will become apparent below, there is no single way to build "the" workflow. Instead, a wide range of tools and approaches exist to optimize the way models are built and deployed. The choice also depends to a large extent on constraints on the project: will everything be run in dedicated datacenters (on premise) or in a cloud-based environment? If the latter, which level will be used, i.e., close to "bare metal" (infrastructure-as-a-service) or will some further services, such as storage or libraries, be included (platform-as-a-service)? Where are the data stored and how can they be accessed? How will the later application, including the model, be deployed? Depending on the requirements imposed by the project at hand, suitable tools can be selected. However, the discussion below provides guidance as to which aspects to consider.

Version Control

Even in highly explorative phases, we should always follow best practices to keep all aspects of our project documented. We should also be able to retrace our steps at each stage of the project to make sure we know which changes lead to a specific outcome.

In general, we must consider three different types of version control systems: one for our code, one for the data, and one for the data science workflow.

Starting with the code, most data science projects are not too different from other types of software development projects, and we should follow the established best practices where applicable, e.g., Huizinga and Kolawa (2007). For example, our code should be well documented and tested by **unit and integration tests**. Coding practices like extreme programming, test-driven development, and clean code help to produce high quality code from early on (Beck, 2002; Beck, 2004; Martin, 2009). When working in

Unit and integration tests

These are two testing methods for software with different foci. Unit tests focus on the correct execution of a piece of software; integration tests aim at ensuring that the software is able to fulfill the task for which it was developed.

Python, packages such as *PyScaffold* (Wilhelm et al., 2018) help to organize the code structure by automatically creating all required directories and helper files to setup code versioning, generation of documentation, testing and management of requirements or dependencies, as well as licenses. The resulting code should be managed via a version control system, such as Git (Chacon & Straub, 2014) or Mercurial (Mackall, 2006). These systems allow multiple developers, who may work from different locations, to easily contribute to the same common code base, add their work to a central repository, note what has changed, resolve conflicts between different submissions, and tag versions when a dedicated milestone has been reached. Even when working alone, it is helpful to use such a version control system to keep track of the evolving work, what has changed, and retain the ability to go back to previous versions as required.

Next, we need to keep track of the data we work with. In order to be able to reproduce results, we need to identify which data were used during model development or (re)training of a specific model. If we train the same model with a different dataset we will generally not be able to reproduce the previous training, particularly if some data were added or removed or some specific data quality issue was addressed in the meantime. Most data handling systems don't provide this functionality. When organizing our data-storage system, we need to add suitable fields to the metadata to be able to implement a rough version control system for the data. For example, we can add a field for the timestamp when the data were first added, additional fields when any modification was made, or version information.

Finally, we need to track all data science experiments, modifications, or extensions we make as we develop our model. With the same version of the code and the same version of the data, we can perform many data science experiments as we tune specific aspects of our model or change the values of hyperparameters, either manually or as part of an automated optimization procedure or cross-validation. The simplest approach is to manually create a new directory structure for each new experiment and store all relevant configuration, model, logs, and results in this directory. This type of recordkeeping is akin to a lab notebook, which scientists have used over the centuries to take detailed notes of their research. Although guidelines about using lab books are available (Pain, 2019; Henry, 2016), specific model versioning systems tailored to the needs of data science experiments help to automate and make it more convenient. In the following, we briefly list and discuss a few examples. Since this dynamic field is developing quickly, the list cannot be exhaustive, nor can it be guaranteed that any solution mentioned here is still viable. Instead, it should serve as a starting point to discuss the individual needs and look for the strengths and weaknesses of individual approaches.

- Weights & Biases (wandb) provides a commercial offer for companies though individual accounts are free. It is easy to integrate as it requires loading a single Python library and provides easy hooks for logging but does require to use the company's Web servers for the actual logging.
- Sacred (Greff et al., 2017) is an open source Python framework that doesn't rely on external services. It aims at adding minimal code to data science experiments and targets early and intermediate stages of the model development where frequent changes are expected. The data scientist specifies "experiments" that contain all

details about the configuration of an individual setting. These experiments are then executed as a "run" during which the relevant parameters are sent to "observers" such as a MongoDB or the file system. Omniboard (Subramanian, 2018) can be used as a Web frontend to investigate the results of the experiments.
- Data Version Control (DVC, 2018) is an open source Python package that is built on top of GIT and uses GIT LFS (Git, 2014) to manage both data science experiments, models, intermediate results, and the data used in these. DVC supports some remote file storage systems such as Amazon S3, Azure, and the Google Cloud store.
- MLflow (Zaharia, 2018) is an open source tool focusing both on logging experiments, their outcomes, packaging machine learning code into reusable forms as well as managing and deploying models. Unlike others, it is built to work with many libraries and provides APIs for Python, R, Java, and REST. The experiments can be recorded in local files, a database, or a remote server.
- RandOpt (Arnold, 2016) is a Python package focusing on machine learning experiment tracking, hyperparameter optimization, and visualization of the results of each experiment.
- Quilt (quiltdata) provides a commercial data versioning tool for Amazon AWS S3.
- Cnvrg.io (cnvrg.io) provides a commercial tool for building machine learning pipelines and tracking experiments.

There are many options to tackle the challenge of data science versioning as the non-exhaustive list above indicates. The most suitable approach depends on the requirements and constraints. An enterprise team working in a big company might want to investigate commercial services, whereas smaller or research groups might favor open source or local installations. The framework used also plays a big role; most systems in this space focus on machine learning modeling with Python.

Building Robust Pipelines

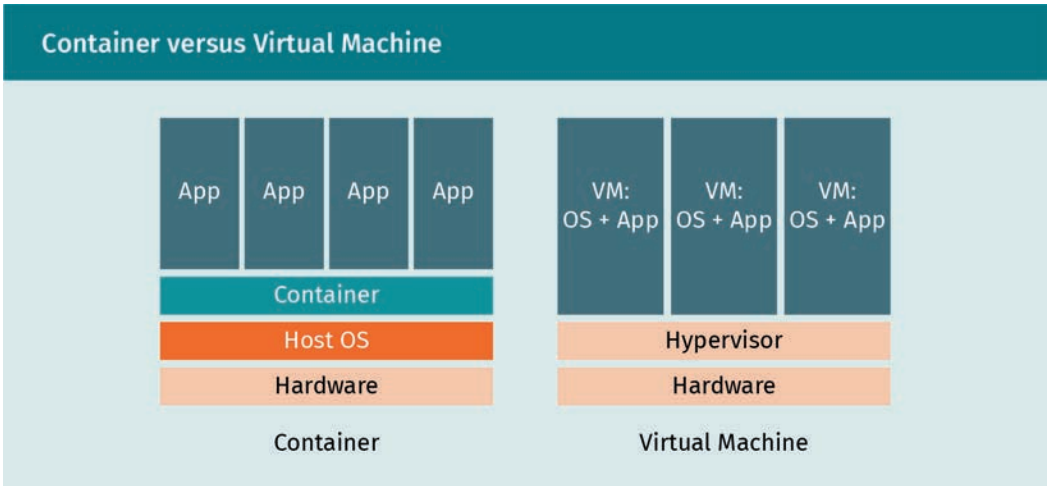
A key part of managing large data science projects is keeping the complexity under control and making sure that the individual components all run both individually and together as a whole. In part, this is similar to general challenges also found in software engineering and software development. For example, the ideas of continuous integration (CI) (Booch, 1993) and continuous delivery (CD) (Chen, 2015) focus on providing software in short delivery cycles and make sure that at each moment a version of the software exists that both completes all required tests and can be deployed. Tools such as Jenkins (2011) provide the necessary infrastructure to support and automate the processes. The Jenkins server or similar tools try to access the relevant source code, execute all defined tests, and try to build and deploy the software. Progress and issues are tracked and visualized allowing for quick action if a bug has been introduced in the recent contributions or if contributions no longer work together, even if they work individually. It is generally advisable to organize the various steps such that each data processing step in handled by a specialized module and multiple modules are joined together to form a **pipeline** that defines all operations on the data.

Pipeline
The concept of pipelines is inspired by those that transport

water or oil. In this case, data "flows" through the pipeline, and various methods and algorithms act on and transform the passing data.

In order to be able to reproduce results and ease the deployment of models across different computers, it is advisable to isolate the working environment from the underlying system. Working in Python, virtual environments allow the decoupling of most of the libraries and packages used during modeling from the versions installed on the computer. In particular, VirtualEnv (Bicking, 2012) and conda (Anaconda, n.d.) can freeze the versions of the libraries used in the modeling. However, these environments are not fully decoupled from the underlying system as some central libraries are still used. Nevertheless, for many practical use cases, they provide a good compromise between ease-of-use and isolation.

Virtual machines (Popek & Goldberg, 1974) allow full decoupling of the environment from the machine on which they are executed. They build a software model of the underlying hardware and simulate everything from CPU, graphics card, storage, and input/output devices. The virtual machine effectively contains both the application software and the operating system. Containers lie between these two approaches. They isolate the applications software along with all relevant dependencies and libraries but rely on a common host operating system to run the containers. The different approach between virtual machines and containers is illustrated below.



Using many of these containers requires using "orchestration" software, which manages the deployment of containers on large computing farms. A popular tool is Kubernetes, inspired by Google's "Borg" tool (Verma et al., 2015). Other tools include AWS Fargate (Hunt, 2017), optimized for the Amazon cloud infrastructure, or Docker Swarm, focusing on the user of docker containers. Kubeflow extends Kubernetes and adds further functionality focused on building machine learning pipelines and deploying machine learning workflows built using docker images. Pachyderm (Pachyderm, 2016) on the other hand, is an open source data science tool centered on building processing pipelines and includes data versioning. It is similar conceptually to working with Git but works with a central server to avoid pulling large files into local file systems. The pipelines are defined using JSON files. Metaflow (Netflix, 2019) is a Python library tightly integrated with Amazon Web Services (AWS) aimed at helping data scientists develop and deploy data science models across the whole stack from model development to scaling operational code on large computing facilities. The idea is to organize all operations as a

"flow" through a directed acyclic graph. Special focus is given to tracking the different versions of different users so that the final graph can be created from individual components similar to how software versions are tagged in a version control system.

Model Persistency and Model Serving

Model persistency and serving is focused on storing a completed and trained model and retrieving it when using it to process new data. This aspect is mainly used in data science projects where the objective centers around production code that needs to run reliably on a compute cluster with high uptime requirements.

The final model code resulting after the model development and training phase can be stored in many formats, such as custom binary data or plain text. Many options exist for storing the resulting model, and some of the data science versioning approaches contain a way to store and identify the model.

Generally, the simplest option is to store the model on a file system accessible to all compute nodes that need to load the model when new data are processed. While this may work well on a small scale, several disadvantages exist. First, some convention has to be defined and maintained to associate the model file with some metadata indicating at least the specific version of the model so it can be identified correctly, especially if a large number of different models are used or models have been improved over time. Moreover, file system access may become very slow once many compute nodes need to access the relevant directory at the same time, and the directory in which the model is stored may become a single point of failure for the entire compute cluster.

Alternatively, the models may be stored in a database, either as a **blob**, an object store (Mesnier et al., 2003), or as a serialized object where the model structure is transformed into byte sequence, essentially a very long string. If the model was developed in Python, the module `pickle` can be used for this. The benefit of storing the model in a suitable database is that it exploits the functionality a database provides to include metadata and handle multiple access requests.

Blob
A blob is a "binary large object" that is stored as a sequence of binary data in the database. Examples of blobs are images, sound, or even executable code. The support for handling these varies from database system to database system.

Model serving is concerned with distributing the trained model to the compute nodes when a new prediction is required. Besides setting up a dedicated database, MLFlow and Kubeflow (discussed earlier) are dedicated tools for this step. If the machine learning model was developed in TensorFlow, Tensorflow-serving (Olston et al., 2017) provides a dedicated mechanism to distribute trained models to the relevant compute nodes.

Summary

While much thought is typically spent on how to build the best model from the perspective of obtaining the best possible description of the data or optimizing the relevant metric, focusing on the process from a technical perspective is equally important, especially if the model is meant to be used in a production environment later on. Quite often, many iterations and improvements are made to a model before it is deemed to be usable for the task. It quickly becomes difficult to keep track of what was changed, when the change was made, and how this change affected the result. Stringent bookkeeping of both code, data, and the results of data science experiments or hyperparameter tuning allow the recording of relevant results. A wide range of tools exist to support the technical aspects of the model development, building robust pipelines and address the question of how to persist the final models and serve them to the compute cluster when new data are processed and predictions have to be made using the previously developed models.

Owing to the diverse settings and constraints in which data science projects are developed, there is no “one-size-fits-all” approach. The most suitable tools have to be combined to support the project.

Knowledge Check

Did you understand this unit?

You can check your understanding by completing the questions for this unit on the learning platform.

Good luck!

Congratulations!

You have now completed the course. After you have completed the knowledge tests on the learning platform, please carry out the evaluation for this course. You will then be eligible to complete your final assessment. Good luck!

Appendix 1

List of References

List of References

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). *Optuna: A next-generation hyperparameter optimization framework*. arXiv. http://arxiv.org/pdf/1907.10902v1

Alhashim, I., & Wonka, P. (2018). High quality monocular depth estimation via transfer learning. arXiv. http://arxiv.org/pdf/1812.11941v2

Anderson, C. (2008, June 23). The end of theory: The data deluge makes the scientific method obsolete. *Wired*. https://www.wired.com/2008/06/pb-theory/

Arnold, S. (2016). RandOpt. Retrieved November 29, 2019, from https://github.com/seba-1511/randopt

Baraniuk, R. B. (2007). Compressive sensing [Lecture Notes]. *IEEE Signal Process. Mag.* 24 (4), 118—121. https://doi.org/10.1109/MSP.2007.4286571

Beck, K. (2002). *Test-driven development: By example*. Addison-Wesley.

Beck, K. (with Andres, C.) (2004). *Extreme programming explained: Embrace change* (2nd ed.). Addison-Wesley.

Benington, H. D. (1956, June 28—29). Symposium on advanced programming methods for digital computer. Office of Naval Research, Department of the Navy. http://worldcatlibraries.org/wcpa/oclc/10794738.

Benington, H. D. (1983). Production of large computer programs. *Annals of History of Computing, 5*(4), 350—361. https://doi.org/10.1109/MAHC.1983.10102

Benjamini, Y., & Hochberg, Y. (1995). Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society: Series B (Methodological), 57*(1), 289—300. https://doi.org/10.1111/j.2517-6161.1995.tb02031.x

Benjamini, Y., & Yekutieli, D. (2001). The control of the false discovery rate in multiple testing under dependency. *The Annals of Statistics, 29*(4), 1165—1188. https://projecteuclid.org/euclid.aos/1013699998

Bicking, I. (2012). VirtualEnv. Retrieved November 29, 2019, from https://github.com/pypa/virtualenv

Bishop, C. M. (1995). Training with noise is equivalent to Tikhonov regularization. *Neural Computation, 7*(1), 108—116. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/bishop-tikhonov-nc-95.pdf

Blacus, V. (2012). Electromagnetic spectrum. Retrieved February 9, 2020, from https://en.wikipedia.org/wiki/File:Electromagnetic-Spectrum.svg

List of References

Booch, G. (1993). *Object-oriented analysis and design with applications*. Benjamin-Cummings.

Booth, E. G., Mount, J., & Viers, J. H. (2006). Hydrologic variability of the Cosumnes River floodplain. *SFEWS, 4*(2). https://doi.org/10.15447/sfews.2006v4iss2art2

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, 144—152.

Box, G. E. P., & Jenkins, G. M. (1970). *Time series analysis: Forecasting and control*. Holden-Day.

Breiman, L. (1996). Bagging predictors. *Machine Learning, 24*, 123—140. https://doi.org/10.1023/A:1018054314350

Breiman, L. (1998). Arcing the edge. *The Annals of Probability, 26*, 1683—1702.

Breiman, L. (2001). Random forests. *Machine Learning, 45*, 5—32. https://doi.org/10.1023/A:1010933404324

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Chapman & Hall/CRC.

Bridle, J. S., & Brown, M. D. (1974). An experimental automatic word-recognition system. (JSRU Report, 1003).

Bryll, R., Gutierrez-Osuna, R., & Quek, F. (2003). Attribute bagging: Improving accuracy of classifier ensembles by using random feature subsets. *Pattern Recognition, 36*(6), 1291—1302. https://doi.org/10.1016/S0031-3203(02)00121-8

Caruana, R., Lou, Y., Gehrke, J., Koch, P., Sturm, M., & Elhadad, N. (2015). Intelligible models for healthcare: Predicting pneumonia risk and hospital 30—day readmission. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1721—1730.

Cauchy, A. (1847). Méthode générale pour la résolution des systemes d'équations simultanées [General method for solving systems of simultaneous equations]. *Comptes Rendus des Seances de l'Academie des Sciences, 25*, 536—538.

Chacon, S., & Straub, B. (2014). *Pro Git: Everything you need to know about Git* (2nd ed). Apress.

Chen, C. H. (Ed.) (1976). *Pattern recognition and artificial intelligence*. Academic Press.

Chen, L. (2015). Continuous delivery: Huge benefits, but challenges too. *IEEE Software, 32*(2), 50—54. https://doi.org/10.1109/MS.2015.27

Christ, M., Braun, N., Neuffer, J., & Kempa-Liehr, A. W. (2018). Time series FeatuRe extraction on basis of scalable hypothesis tests (tsfresh—A Python package). *Neurocomputing, 307*, 72—77. https://doi.org/10.1016/j.neucom.2018.03.067

Cohen, S., Ruppin, E., & Dror, G. (2005). Feature selection based on the Shapley value. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers.

Crowdflower. (2016). Data Science Report. https://visit.figure-eight.com/rs/416-ZBE-142/images/CrowdFlower_DataScienceReport_2016.pdf

Davis, S., & Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions Acoustics, Speech, and Signal Processing, 28*(4), 357—366. https://doi.org/10.1109/TASSP.1980.1163420

DenisBoigelot. (2011). Correlation examples 2. Retrieved February 9, 2020, from https://commons.wikimedia.org/wiki/File:Correlation_examples2.svg

Devijver, P. A., & Kittler, J. (1982). *Pattern recognition: A statistical approach*. Prentice-Hall.

Duignan, B. (1998). Occam's razor. In *Britannica Online*. Retrieved November 27, 2019, from https://www.britannica.com/topic/Occams-razor

DVC. (2018). Data Version Control. Retrieved November 29, 2019, from https://github.com/iterative/dvc

FeatureLabs. (n.d.). Featuretools. Retrieved October 31, 2019, from https://www.featuretools.com/

Feindt, M., & Kerzel, U. (2006). The NeuroBayes neural network package. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 559*(1), 190—194. https://doi.org/10.1016/j.nima.2005.11.166

Fisher, A., Rudin, C., & Dominici, F. (2019). All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *Journal of Machine Learning and Research, 20*, 1—81. http://www.jmlr.org/papers/volume20/18-760/18-760.pdf

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics, 7*(2), 179—188. https://doi.org/10.1111/j.1469-1809.1936.tb02137.x

Fraunhofer IEE. (2018). Windspeed in Germany's Northern and Baltic Sea (FINO). Retrieved October 29, 2019, from http://windmonitor.iee.fraunhofer.de/windmonitor_de/4_Offshore/3_externe_Bedingungen/2_Windbedingungen/

List of References

Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation, 121*(2), 256—285.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences, 55*(1), 119—139.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, *29*(5), 1189—1232.

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis, 38*(4), 367—378.

Gartner. (n.d.). Big data. In *Gartner Glossary*. Retrieved March 27, 2020, from https://www.gartner.com/en/information-technology/glossary/big-data

Geisser, S. (1993). *Predictive inference: An introduction*. CRC Press.

Git. (2014). GIT LFS. Retrieved November 29, 2019, from https://github.com/git-lfs/git-lfs/tree/master/docs

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). *Generative adversarial networks*. arXiv. http://arxiv.org/pdf/1406.2661v1.

Google Research. (2018). *What If...you could inspect a machine learning model, with minimal coding required?* Retrieved November 18, 2019, from https://pair-code.github.io/what-if-tool

Gould, P. (1981). Letting the data speak for themselves. *Annals of the Association of American Geographers, 71*(2), 166—176. https://doi.org/10.1111/j.1467-8306.1981.tb01346.x

Greff, K., Klein, A., Chovanec, M., Hutter, F., & Schmidhuber, J. (2017). The sacred infrastructure for computational research. *Proceedings of the 16th Python in Science Conference*, 49—56. http://conference.scipy.org/proceedings/scipy2017/pdfs/klaus_greff.pdf

Hastie, T. J., & Tibshirani, R. J. (1990). *Generalised additive models*. Chapman and Hall/CRC.

Henderson, R., & Rothe, R. (2017). Picasso: A modular framework for visualizing the learning process of neural network image classifiers. *Journal of Open Research Software, 5*(1), 22. https://doi.org/10.5334/jors.178

Henry, M. (2016). *Consider the lab notebook*. Harvard AIDS Initiative. Retrieved November 29, 2019, from https://aids.harvard.edu/consider-the-lab-notebook/

Ho, T. K. (1995). Random decision forests. *Proceedings of the Third International Conference on Document Analysis and Recognition*, 278—282.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 20*(8), 832—844. https://doi.org/10.1109/34.709601

Hofherr, T., & Kunz, M. (2010). Extreme wind climatology of winter storms in Germany. *Climate Research, 41*(2), 105—123. https://doi.org/10.3354/cr00844

Holt, C. C. (2004). Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting, 20*(1), 5—10. https://doi.org/10.1016/j.ijforecast.2003.09.015

Huizinga, D., & Kolawa, A. (2007). *Automated defect prevention: Best practices in software management*. Wiley-Interscience.

Hunt, R. (2017, November 29). *Introducing AWS Fargate—Run containers without managing infrastructure.* AWS. Retrieved November 29, 2019, from https://aws.amazon.com/blogs/aws/aws-fargate/

IBM. (2012). CRISP-DM in IBM SPSS Modeler. Retrieved October 25, 2019, from https://www.ibm.com/support/knowledgecenter/en/SS3RA7_15.0.0/com.ibm.spss.crispdm.help/crisp_using_in_clem.htm

Itti, L., Koch, C., & Niebur, E. (1998). A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 20*(11), 1254—1259. https://doi.org/10.1109/34.730558

Jacynycz. (2017). Espectrogramas. Retrieved February 9, 2020, from https://commons.wikimedia.org/wiki/File:Espectrogramas.png

Jenkins. (2011). Jenkins. Retrieved November 29, 2019, from https://github.com/jenkinsci/jenkins

Jensen, K. (2012). CRISP-DM process diagram. Retrieved February 9, 2020, from https://commons.wikimedia.org/wiki/File:CRISP-DM_Process_Diagram.png

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *International Joint Conference on Artificial Intelligence,* Montreal, Canada. http://ai.stanford.edu/~ronnyk/accEst.pdf

Laney, D. (2001). *3D data management: Controlling data volume, velocity, and variety.* Gartner Group. http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf

Liu, H., & Setiono, R. (1995, November 5—8). Chi2: feature selection and discretization of numeric attributes. *Proceedings of the 7th International Conference on Tools with Artificial Intelligence*, Herndon, VA, USA, 388—391.

List of References

Lou, Y., Caruana, R., Gehrke, J.M., & Hooker, G. (2013, August). *Accurate intelligible models with pairwise interactions* [Poster presentation]. Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, IL, USA. https://dl.acm.org/doi/10.1145/2487575.2487579

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *31st Conference on Neural Information Processing Systems,* Long Beach, CA, USA. http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

Mackall, M. (2006, July 19—22). Towards a better SCM: Revlog and mercurial. *Proceedings of the Linux Symposium*, Ottowa, Ontario, Canada. vol. 2. https://www.kernel.org/doc/ols/2006/ols2006v2-pages-91-98.pdf

Martin, R. C. (2009). *Clean code: A handbook of agile software craftsmanship*. Prentice Hall.

Mesnier, M., Ganger, G. R., & Riedel, E. (2003). Object-based storage. *IEEE Communications Magazine, 41*(8), 84—90. https://doi.org/10.1109/MCOM.2003.1222722

Micci-Barreca, D. (2001). A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *ACM SIGKDD Explorations Newsletter, 3*(1), 27—32. https://doi.org/10.1145/507533.507538

Microsoft (2016a). *Data Science Utilities from Microsoft*. GitHub. Retrieved October 28, 2019, from https://github.com/Azure/Azure-TDSP-Utilities, updated on 7/31/2018

Microsoft. (2016b). *TDSP project structure, and documents and artifact templates*. GitHub. Retrieved October 28, 2019, from https://github.com/Azure/Azure-TDSP-Project-Template

Microsoft. (2017). *Team Data Science Process Documentation*. Microsoft Azure. Retrieved October 25, 2019, from https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process

Molnar, C. (2019). *Interpretable machine learning: A guide for making black box models interpretable*. Bookdown.

Mordvintsev, A., Olah, C., & Tyka, M. (2015a). *DeepDream—a code example for visualizing neural networks*. Google AI Blog. Retrieved November 11, 2019 from https://ai.googleblog.com/2015/07/deepdream-code-example-for-visualizing.html

Mordvintsev, A., Olah, C., & Tyka, M. (2015b). *Inceptionism: Going deeper into neural networks*. Google AI Blog. Retrieved November 18, 2019 from https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html

NCR Danmark A/S. (1997—1998). *CRoss-Industry Standard Process for Data Mining* [Grant agreement ID: 25959]. European Commission. https://cordis.europa.eu/project/rcn/43348/factsheet/en

Nelder, J. A., & Wedderburn, R. W. M. (1972). Generalized Linear Models. *Journal of the Royal Statistical Society. Series A (General), 135*(3), 370—384. https://doi.org/10.2307/2344614.

Netflix. (2019). *MetaFlow*. Retrieved December 10, 2019 from https://metaflow.org/

Nori, H., Jenkins, S., Koch, P., & Caruana, R. (2019). InterpretML: A unified framework for machine learning interpretability. http://arxiv.org/pdf/1909.09223v1.

Ohno, T. (1988). *Toyota production system: Beyond large-scale production*. Productivity Press.

Olegalexandrov. (2004). Gradient descent. Retrieved February 9, 2020, from https://commons.wikimedia.org/wiki/File:Gradient_descent.svg

Olston, C., Fiedel, N., Gorovoy, K., Harmsen, J., Lao, L., Li, Fangwei, Rajashekar, V., Ramesh, S. & Soyke, J. (2017). TensorFlow-Serving: Flexible, high-performance ML serving. http://arxiv.org/pdf/1712.06139v2.

Pace, R. K., & Barry, R. (1997). Sparse spatial autoregressions. *Statistics & Probability Letters 33*(3), 291—297. https://doi.org/10.1016/S0167-7152(96)00140-X.

Pace, R. K., & Barry, R. (1999). California Housing [Dataset]. http://lib.stat.cmu.edu/datasets/houses.zip

Pachyderm. (2016). Pachyderm. Retrieved November 29, 2019, from https://github.com/pachyderm/pachyderm

Pain, E. (2019). How to keep a lab notebook. *Science*. https://doi.org/10.1126/science.caredit.aaz3678

Pearson, K. (1894). Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 185*, 71—110. https://doi.org/10.1098/rsta.1894.0003

Pearson, K. (1895). VII. Note on regression and inheritance in the case of two parents. *Proceedings of The Royal Society of London, 58*(347—352), 240—242. https://doi.org/10.1098/rspl.1895.0041

Piatetsky-Shapiro, G. (1990). Knowledge discovery in real databases: A report on the IJCAI-89 workshop. *AI Magazine, 11*(4), 68—70. https://doi.org/10.1609/aimag.v11i4.873

Popek, G. J., & Goldberg, R. P. (1974). Formal requirements for virtualizable third generation architectures. *Communications of the ACM, 17*(7), 412—421. https://doi.org/10.1145/361011.361073

Pries, K. H., & Quigley, J. M. (2011). *Scrum project management*. CRC Press.

List of References

Redman, T. C. (2004, April 29). Data doesn't speak for itself. *Harvard Business Review.* https://hbr.org/2014/04/data-doesnt-speak-for-itself

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?": Explaining the predictions of any classifier. http://arxiv.org/pdf/1602.04938v3.

Romary. (2006). Courbe niveau. Retrieved February 9, 2020, from https://commons.wikimedia.org/wiki/File:Courbe_niveau.svg

Roth, A. E. (Ed.). (1988). *The Shapley value: Essays in honor of Lloyd S. Shapley*. Cambridge University Press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*, 533—536. https://doi.org/10.1038/323533a0

Santosa, F., & Symes, W. W. (1986). Linear inversion of band-limited reflection seismograms. *SIAM Journal on Scientific and Statistical Computing, 7*(4), 1307—1330. https://doi.org/10.1137/0907087

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning, 5*, 197—227. https://doi.org/10.1007/BF00116037

Seamen's Church Institute. (1915, November). *The lookout*, vol. 6—7. Seamen's Church Institute Archive.

Shapley, L. S. (1953). A value for n-person games. In H. Kuhn & A. W. Tucker (Eds.). *Contributions to the Theory of Games (AM-23) vol.* 2, 307—317.

Shearer, C. (2000). The CRISP-DM model: The new blueprint for data mining. *Journal of Data Warehousing, 5*, 13—22.

Simonyan, K., Vedaldi, A., & Zisserman, A. (2013). *Deep inside convolutional networks: Visualising image classification models and saliency maps*. arXiv. http://arxiv.org/pdf/1312.6034v2

Sloman, A. (2009). Short history of the clementine data-mining system. Retrieved October, 25 2019, from https://www.cs.bham.ac.uk/research/projects/poplog/isl-docs/clementine.html

Smith, B. T. (2007). *A philosophy of information (information is the power that drives and controls us all).* Trafford Publishing.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research, 15*, 1929—1958. http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Stevens, S. S., Volkmann, J., & Newman, E. B. (1937). A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America,* 8, 185—190. https://doi.org/10.1121/1.1915893

Subramanian, V. (2018). Omniboard. Retrieved November 29, 2019, from https://github.com/vivekratnavel/omniboard

Thorburn, W. M. (1918). The myth of Occam's razor. *Mind, XXVII*(3), 345—353. D https://doi.org/10.1093/mind/XXVII.3.345

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological), 58*(1), 267—288. www.jstor.org/stable/2346178

Tibshirani, R. (1997). The lasso method for variable selection in the cox model. *Statistics in Medicine, 16*, 385—395. http://statweb.stanford.edu/~tibs/lasso/fulltext.pdf

Tikhonov, A. N. (1943). On the stability of inverse problems. *Doklady Akademii Nauk SSSR, 39*(5), 195—198.

Tikhonov, A. N. (1963). Solution of incorrectly formaulated problems and the regularization method. *Soviet Mathematics Doklady*, 4, 1035—1038.

Tikhonov, A. N., & Arsenin, V. Y. (1977). *Solutions of ill-posed problems*. John Wiley & Sons.

Tikhonov, A. N., Goncharsky, A. V., Stepanov, V. V., & Yagola, A. G. (1995). *Numerical methods for the solution of ill-posed problems*. Kluwer Academic.

Tikhonov, A. N., Leonov, A. S., & Yagola, A. G. (1998). *Nonlinear ill-posed problems*. Springer Netherlands.

Verma, A., Pedrose, L., Korupolu, M., Oppenheimer, D., Tune, E., & Wilkes, J. (2015, April 21—24). Large-scale cluster management at Google with Borg. *EuroSys'15*, Bordeaux, France. http://dx.doi.org/10.1145/2741948.2741964

Wachter, S., Mittelstadt, B., & Russell, C. (2018). Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harvard Journal of Law & Technology*, *31*(2). https://jolt.law.harvard.edu/assets/articlePDFs/v31/Counterfactual-Explanations-without-Opening-the-Black-Box-Sandra-Wachter-et-al.pdf

Wachter-Boettcher, S. (2017). *Technically wrong: Sexist apps, biased algorithms, and other threats of toxic tech*. W. W. Norton & Company.

Werbos, P. J. (1994). *The roots of backpropagation: From ordered derivatives to neural networks and political forecasting*. Wiley-Interscience.

List of References

Wilhelm, F., Wick, F., Peters, H., Mühlbauer, P., Rathgeber, F., Schmücker, E., Werner, T., Gethmann, J., Usher, W., Bravalheri, A., Hilton, D., Aguiar, P., Lau, V. C., Podmazo, R., Leni, J., Sottile, A., Häcker, H., Pendleton, N., & Vandenberg, J. (2018). Python project template generator with batteries included. Retrieved November 29, 2019, from https://github.com/pyscaffold/pyscaffold/

Winters, P. R. (1960). Forecasting Sales by Exponentially Weighted Moving Averages. *Management Science, 6*(3), 324—342. http://dx.doi.org/10.1287/mnsc.6.3.324

Zaeri-Amirani, M., Afghah, F., & Mousavi, S. (2018). *A feature selection method ba sed on Shapley value to false alarm reduction in ICUs, a genetic-algorithm approach*. arXiv. http://arxiv.org/pdf/1804.11196v1

Zaharia, M. (2018, June 5). *Introducing MLflow: an open source machine learning platform*. Databricks. Retrieved November 29, 2019, from https://databricks.com/blog/2018/06/05/introducing-mlflow-an-open-source-machine-learning-platform.html

Zeiler, M. D., & Fergus, R. (2013). *Visualizing and understanding convolutional networks*. arXiv. http://arxiv.org/pdf/1311.2901v3

Zeiler, M. D., Taylor, G. W., & Fergus, R. (2011, November 6—13). Adaptive deconvolutional networks for mid and high level feature learning. *International Conference on Computer Vision (ICCV)*, Barcelona, Spain. https://doi.org/10.1109/ICCV.2011.6126474

Zhao, Y., Nasrullah, Z., & Li, Z. (2019). PyOD: A Python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20, 1—7.

Zhou, Z.-H. (2012). *Ensemble methods: Foundations and algorithms*. Chapman & Hall/CRC.

Zhu, J. (2014). *Data envelopment analysis: Let the data speak for themselves*. CreateSpace Independent Publishing Platform.

Appendix 2

List of Tables and Figures

List of Tables and Figures

CRISP-DM Reference Model
Source: Jensen, 2012. CC BY-SA 3.0

House Value
Source: Author.

Comparison of Average Number of Rooms
Source: Author.

Scatterplot Matrix of All Variables
Source: Author.

Latitude versus Longitude versus House Price
Source: Author.

Output of a Linear Regression
Source: Author.

Data Science Lifecycle
Source: Author, based on Microsoft, 2020. Used with permission.

Structured Data (Sales Records)
Source: Author.

Structured Data (Missing Information)
Source: Author.

Structured Data (Non-Conforming)
Source: Author.

Structured Data Example (Outlier)
Source: Author.

Skewed Distribution of Variable X
Source: Author.

List of Tables and Figures

Distribution of Coastal Wind Speeds in Germany
Source: Fraunhofer IEE, 2018. Used with permission.

Maximum Wind Speed in Germany
Source: Hofherr & Kunz, 2010. Used with permission.

Structured Data
Source: Author.

One Hot Encoding
Source: Author.

Electromagnetic Spectrum
Source: Blacus, 2012. CC BY-SA 3.0

X-Ray Image
Source: Stevedangers, Getty Images.

Infrared Image
Source: Marccophoto, Getty Images.

Urban Traffic
Source: Peeterv, Getty Images.

Frequency Spectrum
Source: Jacynycz, 2017. CC BY-SA 4.0

“Common Cold” Search
Source: Author, based on Google Trends, 2019.

Time Series Forecast
Source: Author.

Decomposition of Time Series Data
Source: Author.

Observables
Source: Author.

Correlation between Variables
Source: DenisBoigelot, 2011. In the public domain.

Correlation between Features in the Iris Dataset
Source: Author.

Contingency Table
Source: Author.

Boosting 1
Source: Author.

Boosting 2
Source: Author.

Boosting 3
Source: Author.

Boosting 4
Source: Author.

Linear Data Science Process
Source: Author.

Predictability of Real Systems
Source: Author.

Cross Validation
Source: Author.

Cross Validation for Auto-Correlated Data
Source: Author.

List of Tables and Figures

Sampling with Replacement
Source: Author.

Data Points
Source: Author.

Linear Regression to the Data Points
Source: Author.

Generating Function for the Data Points (Truth)
Source: Author.

Regularized Linear Regression
Source: Author.

Illustration Gradient Descent
Source: Romary, 2006. CC BY-SA 3.0

Illustration of the Optimization Path in Gradient Descent
Source: Olegalexandrov, 2004. In the public domain.

Example Neural Network Architecture Prior to Applying Dropout
Source: Author.

Neural Network Architecture after Applying Dropout
Source: Author.

Low Degree Polynomial Fit to the Data
Source: Author.

Container versus Virtual Machine
Source: Author.

IU Internationale Hochschule GmbH
IU International University of Applied Sciences
Juri-Gagarin-Ring 152
D-99084 Erfurt

Mailing address:
Albert-Proeller-Straße 15-19
D-86675 Buchdorf