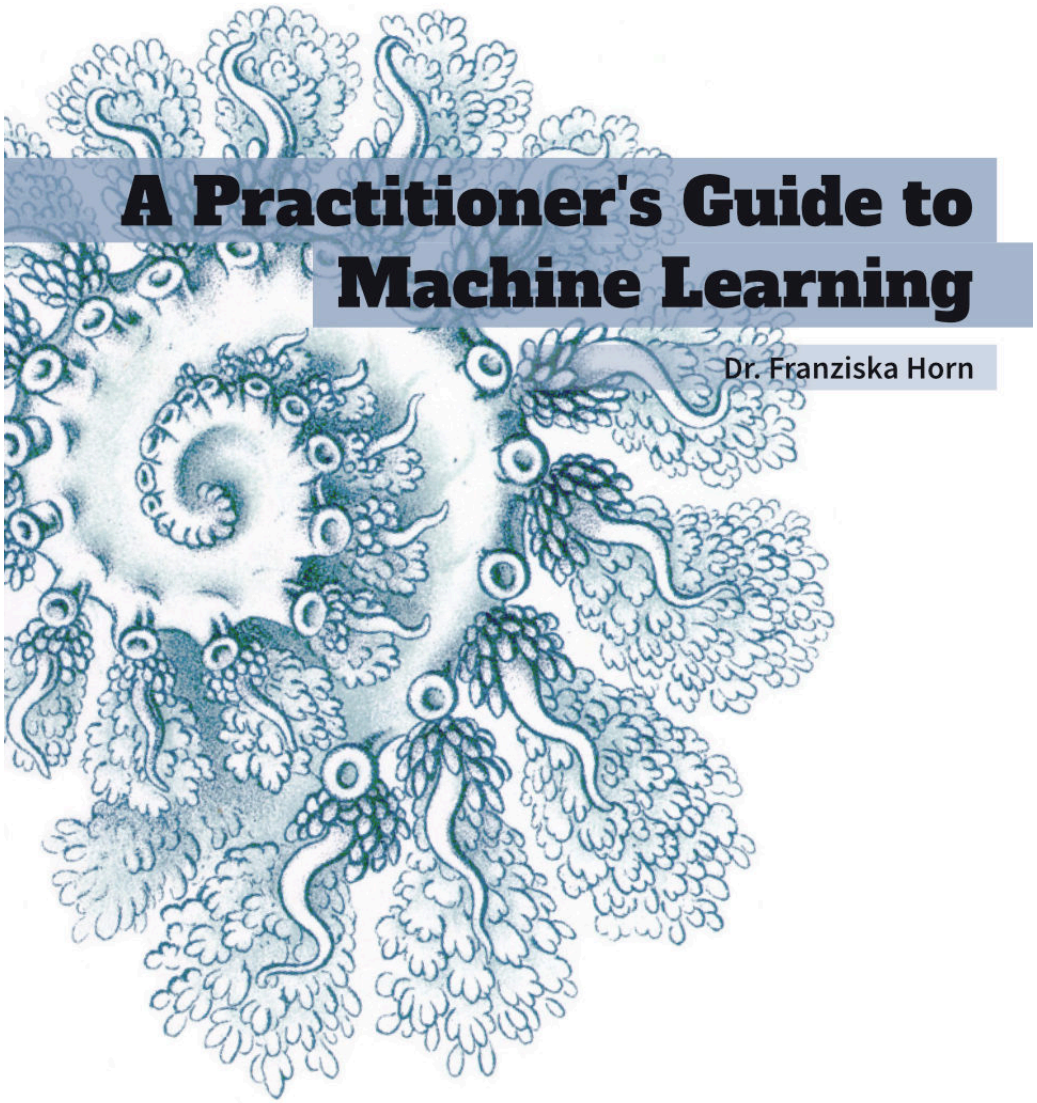


A Practitioner's Guide to Machine Learning

Dr. Franziska Horn

2024-12-13

Table of contents

Preface **1**

Introduction **3**
ML is everywhere! . . . 3
ML history: Why now? . . . 10

The Basics **15**
Data is the new oil!? . . . 15
What is ML? . . . 16
How do machines "learn"? . . . 21
ML use cases . . . 29
Solving problems with ML . . . 41

Data Analysis & Preprocessing **53**
Data Analysis . . . 53
Garbage in, Garbage out! . . . 61
Data Preprocessing . . . 65

Deep Learning **67**
Neural Networks . . . 67

Avoiding Common Pitfalls **73**
[Pitfall #1] Deceptive model evaluation . . . 74
[Pitfall #2] Model does not generalize . . . 75
[Pitfall #3] Model abuses spurious correlations . . . 77
[Pitfall #4] Model discriminates . . . 79
[Pitfall #5] Data & Concept Drifts . . . 84

Conclusion **89**
AI Transformation of a Company . . . 91

Preface

Why read this book?

There are a lot of machine learning (ML) resources out there. Many of them either targeted at students or researchers and rather heavy on the mathematical theory, or others in the form of tutorials, focusing on the concrete implementation and application of some ML algorithm to a specific problem. This book tries to find a middle ground between both the theoretical background, which I have studied in depth while completing my PhD in machine learning at the TU Berlin, Germany, and the practical applications of these algorithms to solve different problems, as I have been doing in the last few years as an independent data science consultant for various firms. This book originated from my experience holding dozens of machine learning seminars and workshops in front of audiences with varying levels of technical and mathematical background.

Questions this book answers:

- Which problems can machine learning (ML) solve?
- How does ML solve these problems, i.e., how do the algorithms work (in theory)?
- How do you actually get this to work in practice and avoid common pitfalls?

This book does **not** explain the latest fancy neural network model that achieves state-of-the-art performance on some specific task. Instead it provides a general intuition for the ideas behind different machine learning algorithms to establish a solid framework that helps you better understand and integrate into a bigger picture what you later read about these specific approaches.

This book exist in two versions:

- The full version, targeted at (about to be) data scientists.
- The condensed version, targeted at a more general audience (also available in German).

The condensed version is written for all audiences, i.e., readers generally interested in ML, who want to understand what is behind the hype and where ML can – or should not – be used. The full version is mainly written for ML practitioners and assumes the reader is familiar with elementary concepts of linear algebra (see also this overview on the mathematical notation used in the book).

If you prefer to work through the book's content in a more structured way as part of a group, you can also enroll in one of my online courses. There, you'll also have the opportunity to discuss questions.

This is still a draft version! Please write me an email or fill out the feedback survey, if you have any suggestions for how this book could be improved!

Enjoy!

Acknowledgments

I would like to thank: Antje Relitz for her feedback & contributions to the original workshop materials, Robin Horn for his feedback & help with the German translation of the book, and Karin Zink for her help with some of the graphics (incl. the book cover[1]).

How to cite

```
@book{horn2021mlpractitioner,
  author = {Horn, Franziska},
  title = {A Practitioner's Guide to Machine Learning},
  year = {2021},
  url = {https://franziskahorn.de/mlbook/},
}
```

[1]Book cover, featuring a drawing of part of a Siphonophorae (a kind of jellyfish) by Ernst Haeckel from his book "Kunstformen der Natur" (1900, Tafel 37; source: www.BioLib.de).

Introduction

This chapter provides some motivating examples illustrating the rise of machine learning (ML).

ML is everywhere!

Machine learning is already used all around us to make our lives more convenient:

Face recognition

Face recognition technology is one of the earliest notable examples of machine learning and computer vision that can nowadays be found in every digital camera and smartphone.

While the algorithms implemented in a camera application are fairly simple and only detect the presence of faces in general to make sure you look your best when the picture is taken, more sophisticated algorithms are also being used by governments and law enforcement in more and more countries to match a detected face to a known person in their biometric databases, for example, to identify criminals. So…smile!?

Figure 1: Source: https://thesocietypages.org/socimages/2008/12/15/nikon-s60-auto-focuses-on-voyeurs-savages-ghosts/ (15.12.2008)

Object recognition (e.g., for self-driving cars)

Another example from the area of computer vision is object recognition or image segmentation in general. This is, for example, used in self-driving cars to make sure they are aware of street signs and pedestrians.

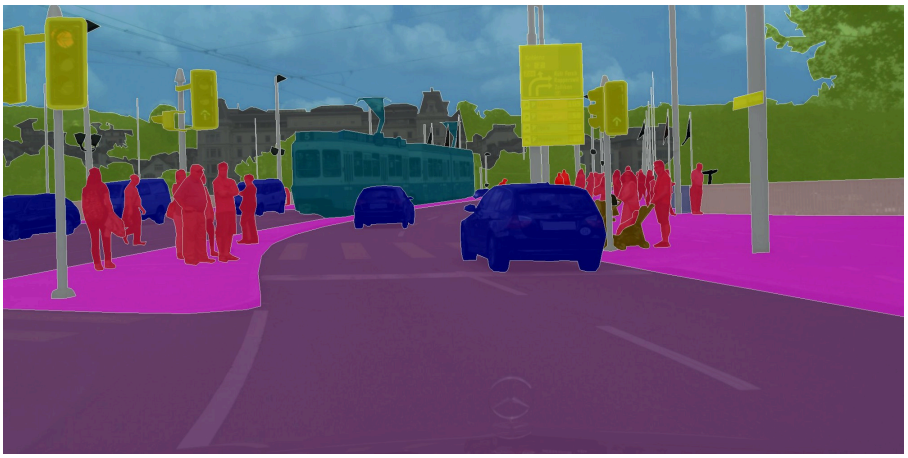


Figure 2: Source: https://medium.com/intro-to-artificial-intelligence/c01eb6eaf9d (16.06.2018)

Analysis of medical images

The last example on image data comes from the application area of medicine: Below you see two images of retinas, i.e., photos taken of the back of someone's eye, based on which it is possible to diagnose a common complication of diabetes that can result in blindness if left untreated.

The diagnostic algorithm to identify the markers of the disease in these images was developed by researchers at Google and achieves the same level of accuracy as human experts in the field. Google had even assembled a team of top specialists to discuss the hardest cases again to get consistent labels for all images, which gave their model an additional performance boost.

Since the equipment to take these images is fairly cheap, this means that with this ML model, expert diagnostic decisions can now be made available to those that might otherwise not have had the means to consult a top specialist.

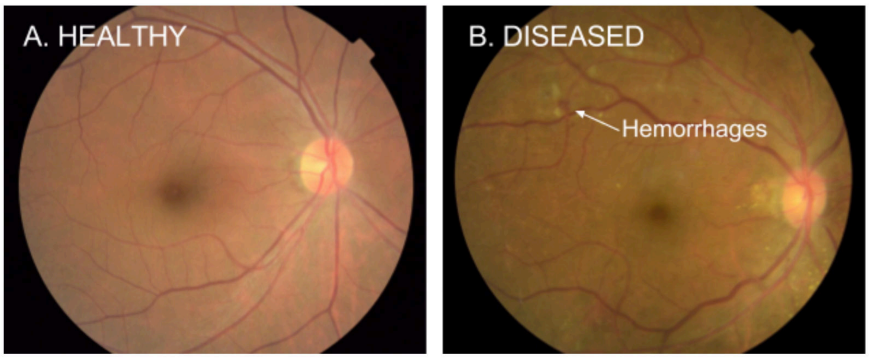


Figure 1. Examples of retinal fundus photographs that are taken to screen for DR. The image on the left is of a healthy retina (A), whereas the image on the right is a retina with referable diabetic retinopathy (B) due a number of hemorrhages (red spots) present.

Figure 3: Source: https://ai.googleblog.com/2016/11/deep-learning-for-detection-of-diabetic.html (29.11.2016)

Conversational agents (i.e., speech recognition…)

Leaving the field of computer vision, now an example from the area of natural language processing (NLP): Conversational agents, like Siri or Alexa, are waiting for commands in many people's homes. While many of the answers they give are still scripted by humans (as in the screenshot below), the real challenge is to understand what the person had actually said in the first place. Speech recognition, i.e., automatically transcribing spoken language into text, is a rather difficult problem, for example, since people speak with different accents and there can be additional background noises.

Figure 4: Screenshot: Siri on macOS (13.12.2018)

Machine translation

Again from the field of NLP: machine translation, i.e., automatically translating text from one language into another.

If you have used Google Translate (shown as an example in the screenshot below) after it was first released in 2006, you were probably often quite disappointed with the results, as the translated sentences read more like the words were just looked up one after another in a dictionary (= statistical machine translation). However, this changed when Google made the switch to a neural network model

to generate the translations 10 years later in 2016: now the translated texts are actually readable and usually require only minor manual corrections, if any.

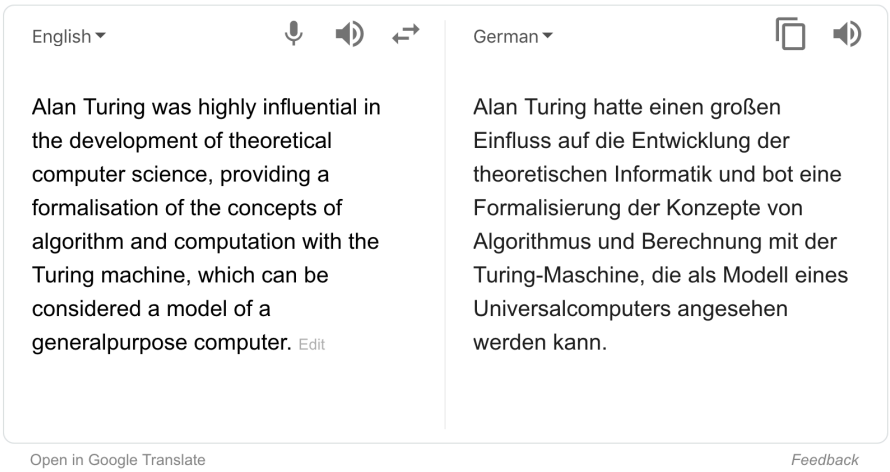


Figure 5: Screenshot: https://translate.google.com/ (13.12.2018)

Recommender systems

Another application area are recommender systems, for example, on e-commerce platforms (like Amazon in the screenshot below) to provide the user with helpful search results and suggestions, and thereby generate revenue for the respective companies. They are also used on social media platforms and by Netflix, YouTube & co to keep you glued to your screen.

Sometimes the provided suggestions might help you find exactly what you were looking for. But especially platforms with uncurated content such as YouTube have also been criticized for fostering, e.g., conspiracy theories through these personalized recommendations. Since this kind of content kept users especially engaged, it was recommended a lot and thereby drove the users further down some rabbit hole instead of also providing perspectives outside one's own information bubble.

But on the upside, the research on recommender systems has also sparked developments in other areas of science, such as methods that recommend drug molecules that fit to the proteins playing a key role in certain diseases to accelerate the search for a cure.

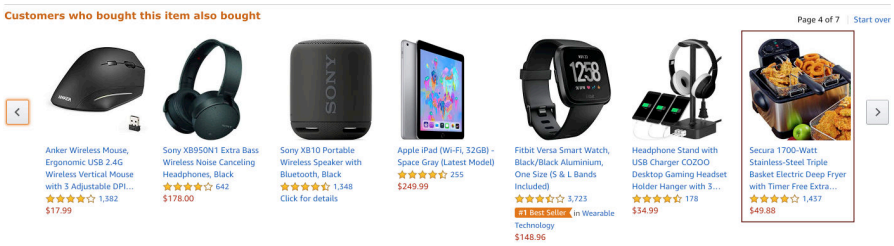


Figure 6: Screenshot: https://www.amazon.com/ (12.12.2018)

Better than humans: AlphaGo

In 2016, DeepMind, a startup subsequently acquired by Google, presented AlphaGo, the first computer program to beat a human Go master.

This was a huge milestone for the AI research community, as Go, with a 19 x 19 playing field, is a lot more complex than chess (8 x 8 tiles and more restrictive movement patterns), and even the most optimistic AI researchers had not expected that a computer could win against a Go master before 2020.

The algorithms used in AlphaGo are from the subfield of reinforcement learning, which we will discuss in more detail later.

Figure 7: Source: https://www.nature.com/nature/volumes/529/issues/7587 (28.01.2016)

Protein folding - solving a 50-year-old challenge

DeepMind presented another major success story in 2020: Their AlphaFold model now estimates the 3D structure of a protein from its raw amino acid sequence as accurately as traditional simulation models.

Proteins often play a key role in diseases. If we know a protein's 3D structure, we can determine which drug molecules can bind to it and thereby identify target structures that should be investigated further to find a cure for the disease.

While exact simulation models to estimate a protein's 3D structure existed for a long time, these were very slow and it often took several days to compute the folding for a single protein. With the new neural network model, the same computation can now be done in a matter of minutes or even seconds, thereby vastly accelerating drug development.

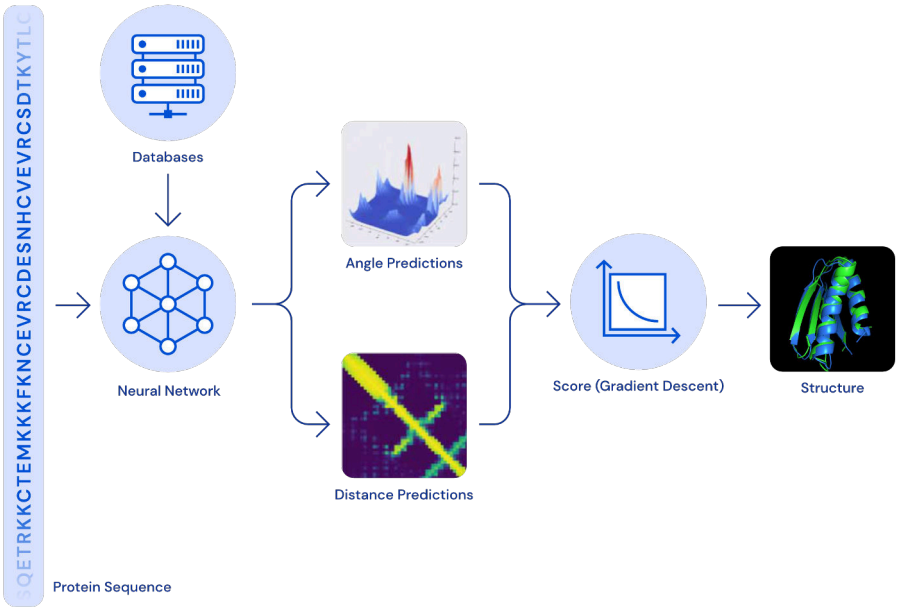


Figure 8: Source: https://deepmind.google/discover/blog/alphafold-using-ai-for-scientific-discovery-2020/ (15.01.2020)

Neural networks getting creative

Lots of fun applications use neural networks to create new content, i.e., perform creative tasks that were previously thought exclusive to humans.

For example, an AI has written a slightly confusing yet hilarious script for a movie, which was then actually produced.

Neural networks are also used to visualize music by combining and transforming images, like in the video below:

https://www.youtube.com/embed/85l961MmY8Y

And you've probably also seen some examples of "Neural Style Transfer" before, a technique that can, for example, be used to make your social media profile picture look like a van Gogh painting:

Figure 9: Source: https://pytorch.org (28.05.2022)

Even stock photos are now basically obsolete, since we can use neural networks to generate images given a textual description:

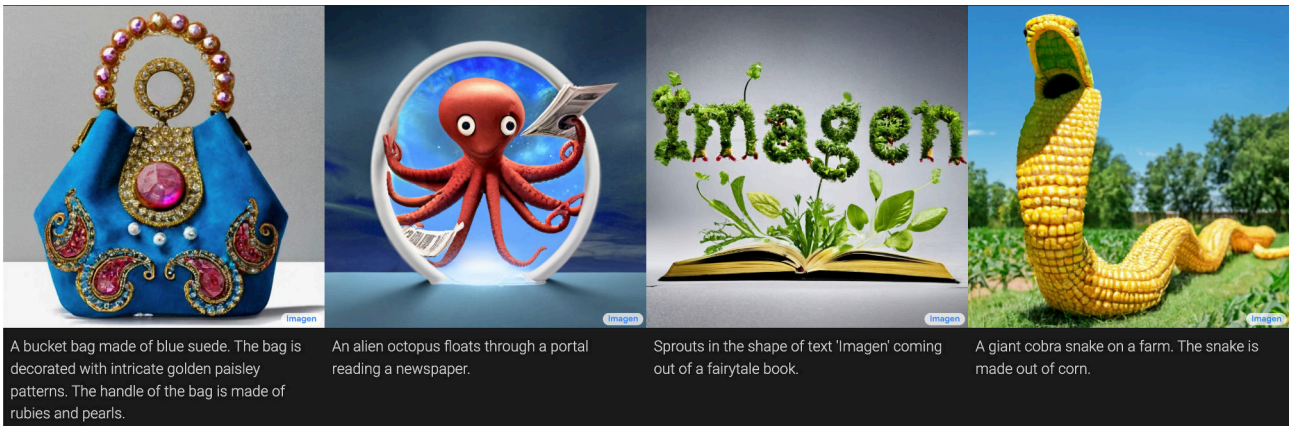


Figure 10: Screenshot: https://imagen.research.google/ (28.05.2022)

And chatbots can hold conversations and provide information in a variety of styles:

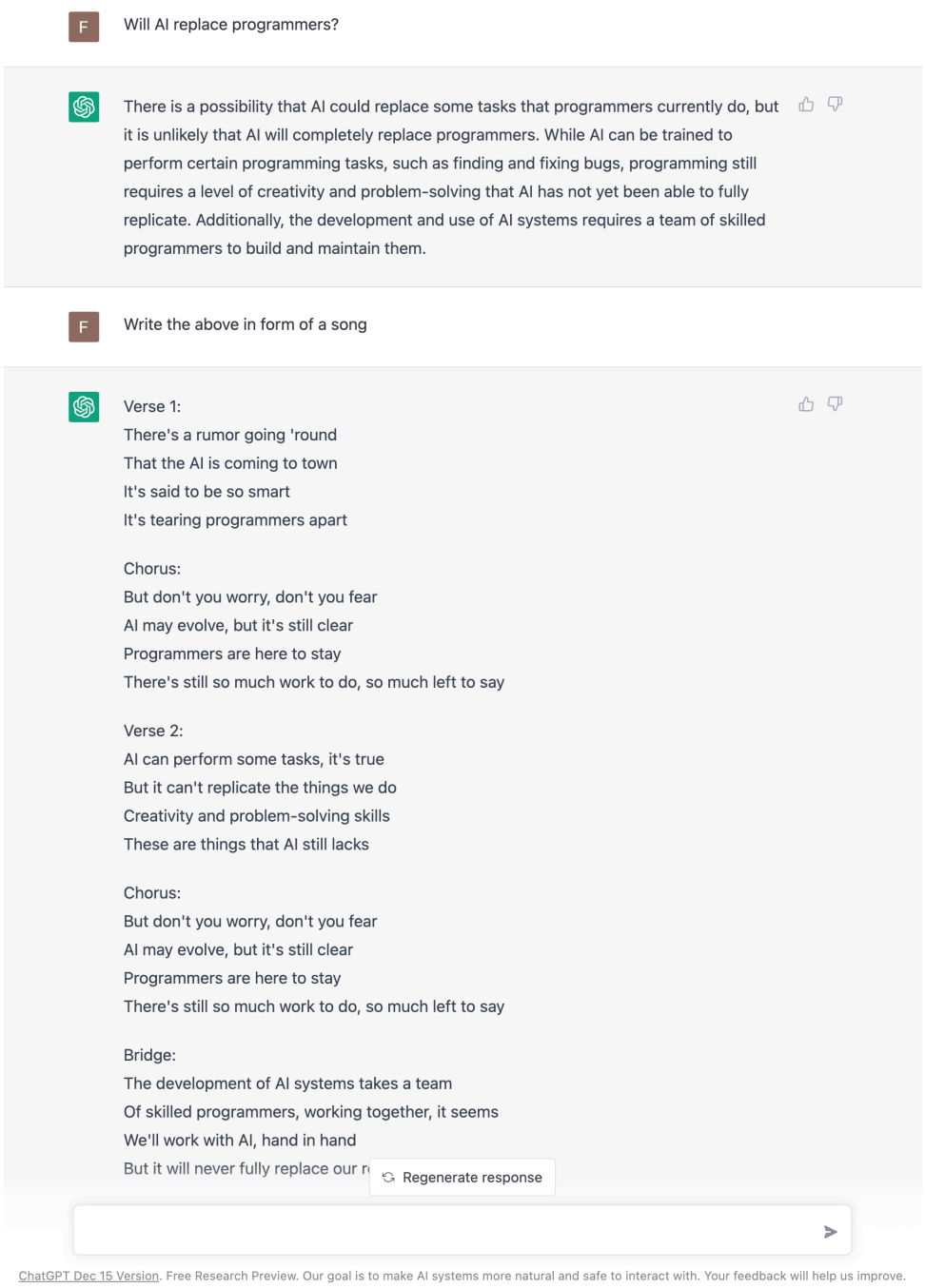


Figure 11: Screenshot: https://chat.openai.com/chat (04.01.2023)

ML history: Why now?

Why is there such a rise in ML applications? Not only in our everyday lives has ML become omnipresent, but also the number of research paper published each year has increased exponentially:

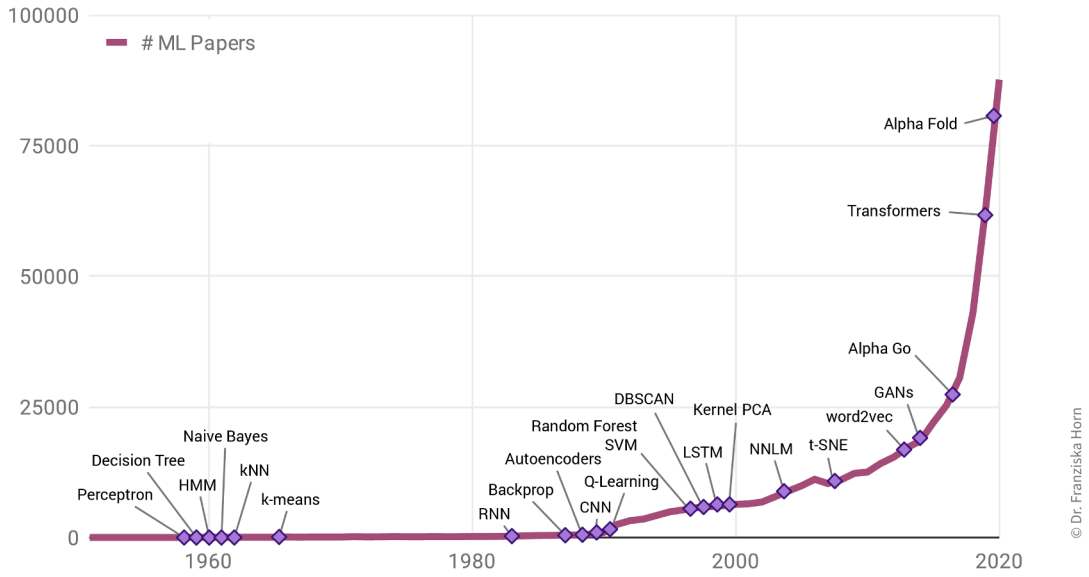


Figure 12: Data Source: https://www.webofknowledge.com/

Interestingly, this is not due to an abundance of groundbreaking theoretical accomplishments in the last few years (indicated as purple diamonds in the plot), but rather many of the algorithms used today were actually developed as far back as the late 50s / early 60s. For example, the perceptron is a precursor of neural networks, which are behind all the examples shown in the last section. Indeed, some of the most important neural network architectures, recurrent neural networks (RNN) and convolutional neural networks (CNN), which provide the foundation for state-of-the-art language and image processing respectively, were developed in the early 80s and 90s. But back then we lacked the computational resources to use them on anything more than small toy datasets.

This is why the rise in ML publications correlates more closely with the number of transistors on CPUs (i.e., the regular processors in normal computers) and GPUs (graphics cards, which parallelize the kinds of computations needed to train neural network models efficiently):

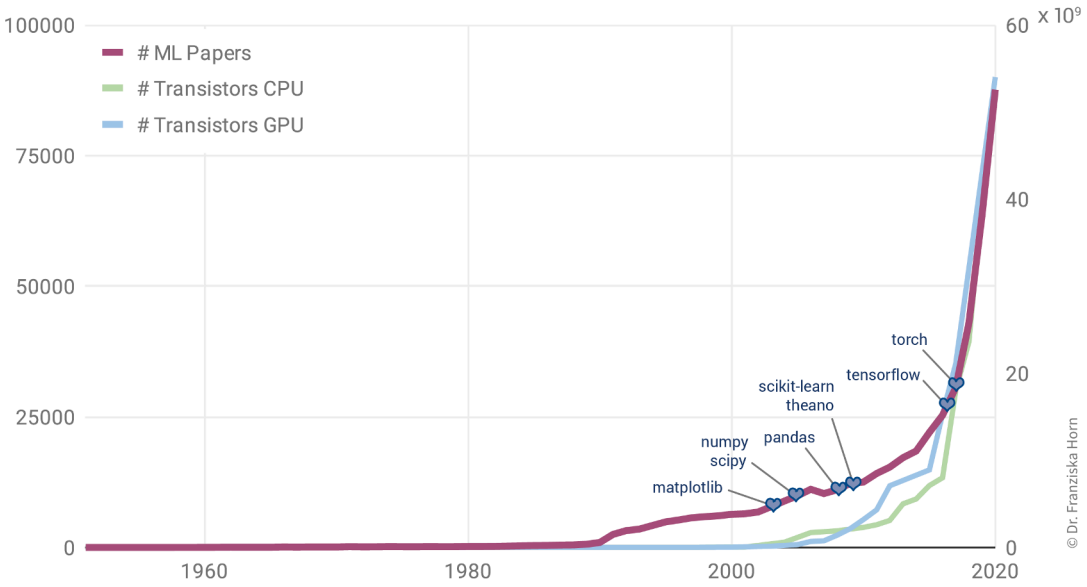
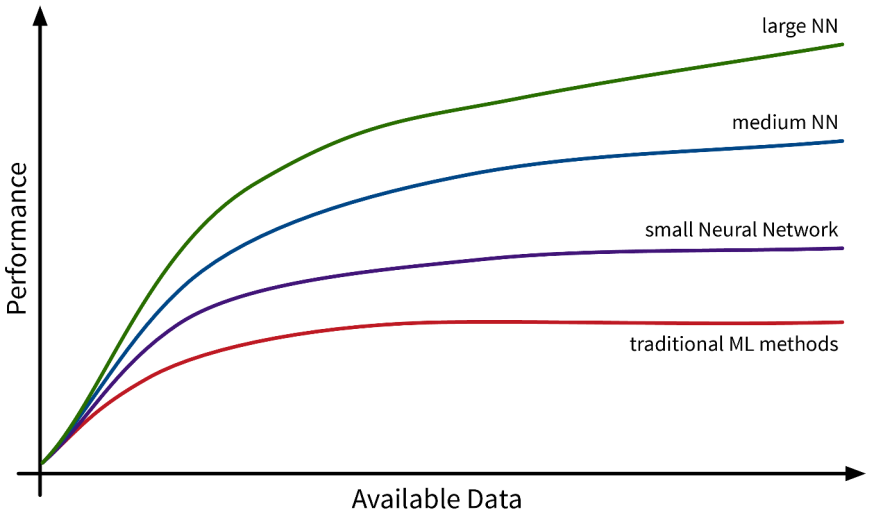


Figure 13: Data Source: https://en.wikipedia.org/wiki/Transistor_count

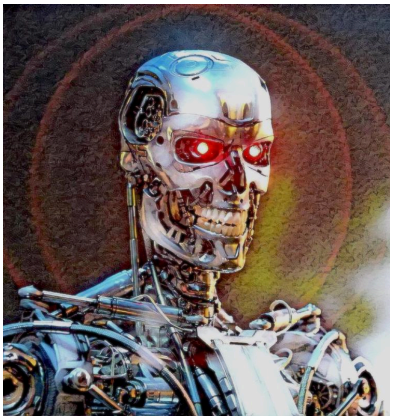
Additionally, the release of many open source libraries, such as scikit-learn (for traditional ML models) and theano, tensorflow, and (py)torch (for the implementation of neural networks), has further facilitated the use of ML algorithms in many different fields.

> **Note**
>
> While these libraries democratize the use of ML, unfortunately, this also brings with it the downside that ML is now often applied without a sound understanding of the theoretical underpinnings of these algorithms or their assumptions about the data. This can result in models that don't show the expected performance and subsequently some (misplaced) disappointment in ML. In the worst case, it can lead to models that discriminate against certain parts of the population, e.g., credit scoring algorithms used by banks that systematically give women loans at higher interest rates than men due to biases encoded in the historical data used to train the models. We'll discuss these kinds of issues in the chapter on avoiding common pitfalls.

Another factor driving the spread of ML is the availability of (digital) data. Companies like Google, Amazon, and Meta have had a head start here, as their business model was built around data from the start, but other companies are starting to catch up. While traditional ML models do not benefit much from all this available data, large neural network models with many degrees of freedom can now show their full potential by learning from all the texts and images posted every day on the Internet:



But we're still far from Artificial **General** Intelligence (AGI)!



An AGI is a hypothetical computer system with human-like cognitive abilities capable of understanding, learning, and performing a **wide range of tasks across diverse domains**. Specifically, an AGI would not only perform specific tasks but also understand and learn from its environment, make decisions autonomously, and **generalize its knowledge to completely new situations**.

Instead, what is used in practice today is **Artificial *Narrow* Intelligence** (ANI): models **explicitly programmed to solve a specific task(s)**, e.g., translate texts from one language to another. But they **can't generalize (on their own) to handle novel tasks**, i.e., a machine translation model will not tomorrow decide that it now also wants to recognize faces in images. Of course, one can combine several individual ANIs into one big program trained to solve multiple different tasks, but this collection of ANIs is still not able to learn (on its own) any new skills beyond these capabilities.

Many AI researchers believe that at least with the currently used approaches to AI (e.g., large language models (LLMs) like ChatGPT from OpenAI), we might never produce a true human-like AGI. Specifically, these AI systems still **lack a general understanding of causality and physical laws like object permanence** – something that even many pets understand.
If you're interested to learn more about the faults of current AI systems, the blog articles by Gary Marcus are highly recommended!

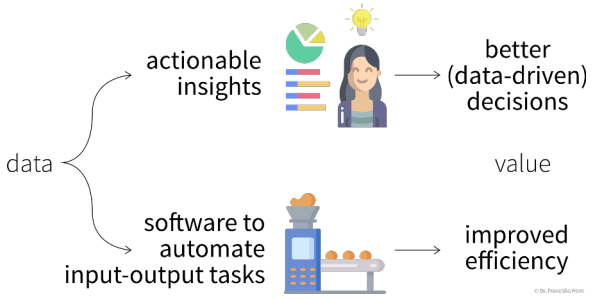
The Basics

This chapter provides a general introduction into what machine learning (ML) actually is and where it can – or should not – be used.

Data is the new oil!?

Let's take a step back. Because it all begins with data. You've probably heard this claim before: "Data is the new oil!". This suggests that data is valuable. But is it?
The reason why oil is considered valuable is because we have important use cases for it: powering our cars, heating our homes, and producing plastics or fertilizers. Similarly, our data is only as valuable as what we make of it. So what can we use data for?

The main use cases belong to one of two categories:



Insights

We can generate insights either through continuous monitoring ("Are we on track?") or a deeper analysis ("What's wrong?").

By visualizing important variables or *Key Performance Indicators* (KPIs) in **reports** or **dashboards**, we increase transparency of the status quo and quantify our progress towards some goal. When a KPI is far from its target value, we can dig deeper into the data with an exploratory data analysis to identify the root cause of the problem and answer questions such as

- Why are we not reaching our goal?
- What should we do next?

However, as we'll discuss in more detail in the section on data analysis, arriving at satisfactory answers is often more art than science .

Automation

As described in the following sections, machine learning models can be used to **automate 'input → output' tasks** otherwise requiring a human (expert). These tasks are usually easy for an (appropriately trained) human, for example:

- Translating texts from one language into another
- Sorting out products with scratches when they pass a checkpoint on the assembly line
- Recommending movies to a friend

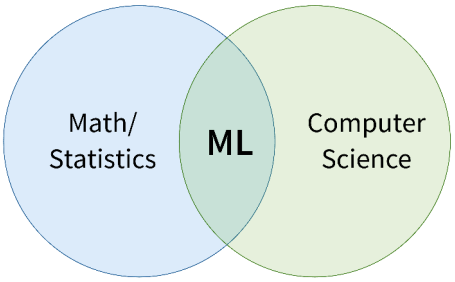
For this to work, the ML models need to be **trained on a lot of historical data** (e.g., texts in both languages, images of products with and without scratches, information about different users and which movies they watched).

The resulting software can then either be used to **automate the task completely** or we can keep a **human in the loop** that can intervene and correct the suggestions made by the model.

What is ML?

OK, now what exactly is this machine learning that is already transforming all of our lives?

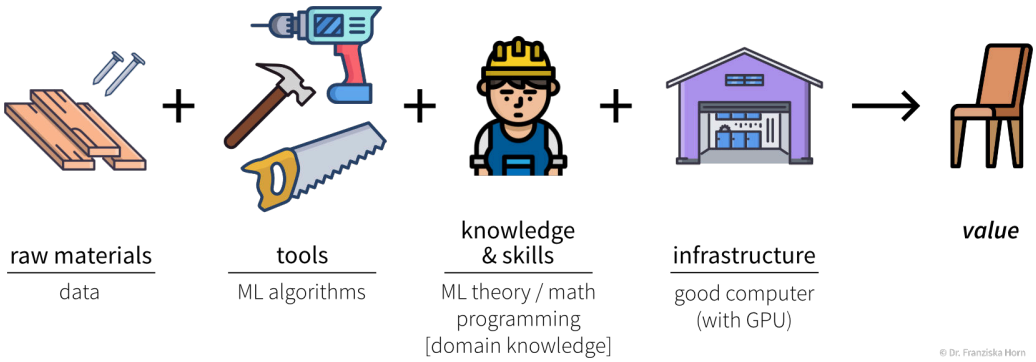
First of all, ML is an area of research in the field of theoretical computer science, i.e., at the intersection of mathematics and computer science:



More specifically, **machine learning** is an **umbrella term for algorithms that recognize patterns and learn rules from data**.

> **Note**
>
> Simply speaking, an **algorithm** can be thought of as a **strategy or recipe for solving a certain kind of problem**. For example, there exist effective algorithms to find the shortest paths between two cities (e.g., used in Google Maps to give directions) or to solve scheduling problems, such as: "Which task should be done first and which task after that to finish all tasks before their respective deadlines and satisfy dependencies between the tasks." Machine learning deals with the subset of algorithms that detect and make use of statistical regularities in a dataset to obtain specific results.

Analogous to the tools used in a traditional manufacturing process to build something, you can think of **ML algorithms as tools to generate value from data**:



In order to successfully apply ML, you should ask yourself some important questions:

- ***What could be valuable?*** For example, this could be a new feature for an existing product, like Face ID as a new way to unlock your phone.
- ***What raw inputs are needed?*** We can't build a wooden chair using only fabric and metal or a few twigs we found in the woods. Similarly, depending on what we want to achieve with ML, we also need the right data (quality & quantity) to apply the algorithms in the first place. This can be especially tricky since in most cases we can't just buy the data we need like wood at a hardware store, but we have to collect it ourselves, i.e., grow our own trees, which can take some time.
- ***Which ML algorithm is the right tool for the task?*** (I.e., which category of ML algorithms produces the type of output we want?)
- Do I or my employees have the ***necessary skills and enough compute power*** to accomplish this in practice?

We can think of the different ML algorithms as our **ML toolbox**:

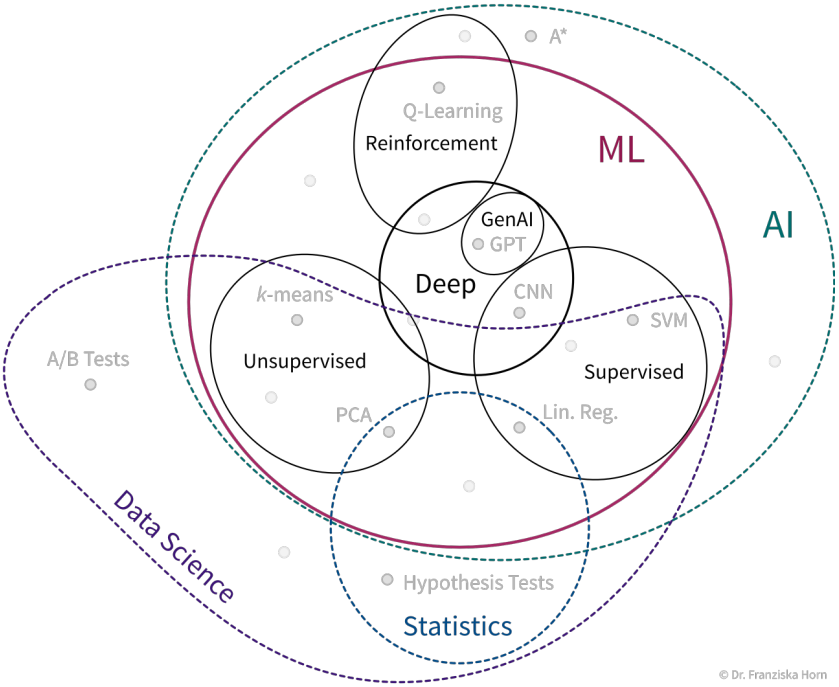


Figure 1: ML itself is a subfield of AI, which is currently the more frequently used buzzword, but all the cool applications (e.g., the examples we've seen in the beginning) actually use ML. Besides ML, AI includes, for example, some search algorithms that were used for building the first chess computers. ML can be divided into three main subfields, unsupervised, supervised, and reinforcement learning. Additionally, the subfield "deep learning" is a buzzword for neural network models and also includes Generative AI (GenAI) models like ChatGPT. Some of the simplest algorithms used in ML, like linear regression or PCA (very similar to factor analysis), are also used by statisticians, who additionally use other tools, like hypothesis tests, which do not learn rules or patterns from data. Finally, most data scientists use many tools from ML and statistics, but they as well use some additional tools like A/B tests, e.g., for collecting data on whether a red or green "buy" button on a website generates more sales, which do not fall into any of the other categories.

ML algorithms solve "input → output" problems

What all of these ML algorithms have in common, is that they solve "input → output" problems like these:

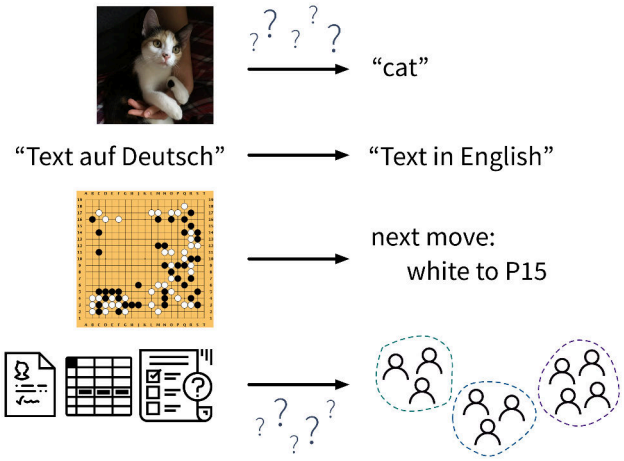


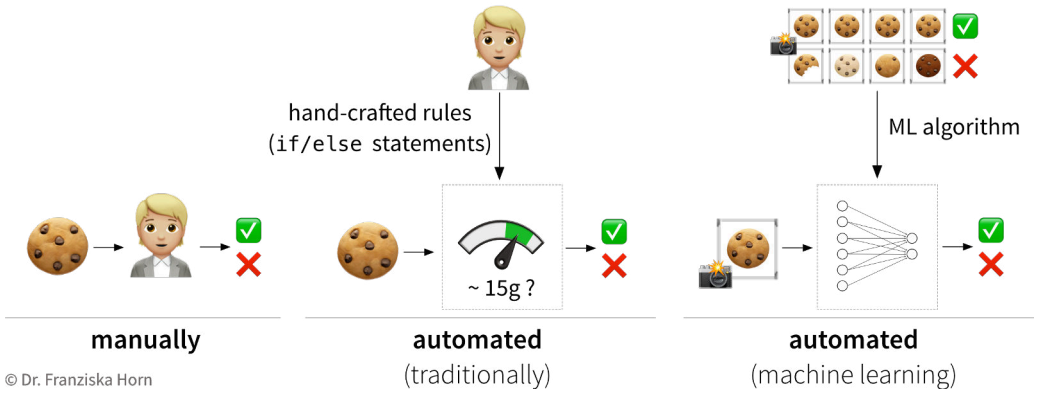
Figure 2: Example "input → output" ML problems: recognizing objects in images; translating text from one language to another; determining a good next move given the current state of a Go board; grouping similar users/customers together based on some information about them like questionnaire answers (known as customer segmentation in marketing, this is used, for example, to target different groups of customers with specific advertisement campaigns on social media).

In the above examples, while a human (expert) could easily produce the correct output given the input (e.g., even a small child can recognize the cat in the first image), humans have a hard time describing *how* they arrived at the correct answer (e.g., how did you know that this is a cat (and not a small dog)? because of the pointy ears? the whiskers?). ML algorithms can **learn such rules from the given data samples**.

ML vs. traditional software

While **traditional software** solutions are used to automate tasks that can be formulated as a fixed, predefined sequence of actions, executed according to some **hard-coded rules** (e.g., "a gate should open *if* an object passes through a photoelectric barrier and 20 seconds later the gate should close again"), machine learning can be used to **automate "input → output" tasks** for which it would otherwise be **difficult to come up with such rules**.

For example, the quality control in a cookie factory is such an "input (cookie) → output (ok/defective)" task: While some broken cookies could be sorted out automatically by checking that each cookie weights around 15g, it would be difficult to formulate rules that reliably catch all possible defects. So either a human could watch the production line to additionally recognize, e.g., over-baked cookies, or one could take pictures of the cookies and use them as input for a machine learning model to recognize the defective cookies:



To solve this problem with ML, first a large dataset needs to be compiled with photos of many good, but also all kinds of defective cookies, including the corresponding annotations, i.e., a label for each picture whether it displays a good or defective cookie (not necessarily specifying the kind of defect). An ML algorithm can then learn to distinguish between good and defective cookies from these examples.

When (not) to use ML

ML is overkill if:

- a manually defined set of rules or mechanistic (white box) model can solve the problem. For example, if in our example cookie factory broken cookies were the only quality problem that ever occurred, then the rule "cookie weight needs to be between 14-16g" would suffice to detect defective cookies. And such a rule is easier to implement as there is no need to collect a large dataset.

ML has great potential when:

- an exact simulation with a mechanistic model takes too long (but can be used to generate a high quality dataset). For example, the AlphaFold model shown in the introduction, which is used to predict the 3D structure of a protein from its amino acid sequence, can be trained on the data generated by the original simulation model used to solve this task before, which is too slow to be applied to a large number of proteins.
- solving a "simple" but hard to explain task that takes a human ~1 second, like recognizing something in an image.

  Use ML to automate repetitive tasks & make expert knowledge available to everyone, e.g., Google's diabetic retinopathy diagnostic model shown in the first section.
  But: success depends on data quality & quantity!
  → Humans are much better at generalizing from a few examples. For example, a doctor can still easily recognize the disease even if the pictures were taken with a slightly different setup that might result, for example, in noisier images. The ML model, on the other hand, needs to be specifically trained for these cases, which means that in the worst case we might need to collect a lot of additional data for this new setup.

ML is your best chance when:

- humans are overwhelmed by very complex, high dimensional data. For example, given an excel spreadsheet with hundreds of columns, a human can't easily recognize any patterns in this sea of numbers. In the worst case, there actually aren't any relationships in the data that could be discovered (maybe we didn't measure all the relevant factors), but if there are, ML will most likely find them.

Caution

Use ML only when occasional errors are acceptable. ML models are typically trained on human-generated data, which is prone to noise since even experts may disagree on certain cases. Additionally, ML models may need to extrapolate, predicting outcomes for new data points that differ from the training data, leading to potential inaccuracies. To minimize errors, keeping a human in the loop to periodically review the predictions made by the ML model can be beneficial.

How do machines "learn"?

How do ML algorithms solve these "input → output" problems, i.e., how do they recognize patterns and learn rules from data?

The set of ML algorithms can be subdivided according to their learning strategy. This is inspired by how humans learn:

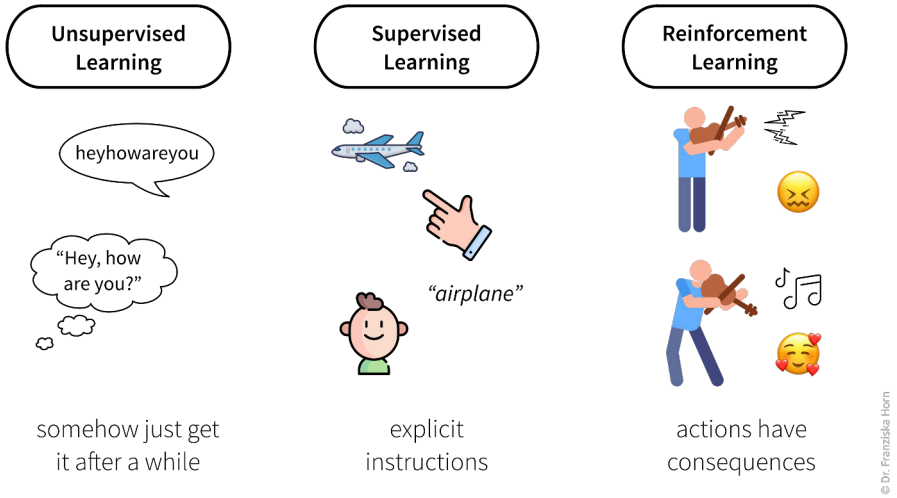


Figure 3: **Unsupervised Learning:** Humans are very good at picking up on statistical regularities in the world without being explicitly instructed to do so. For example, have you ever noticed that we don't … make … pauses … between … words when we speak? Yet kids still intuitively learn which syllables make up a word and where this word ends and the next one begins. This is possible, because the syllables in a single word always occur in this specific combination, while this word can then be followed by many different words, starting with many different syllables. This means, simply by hearing lots of spoken text, we pick up on the conditional probability distributions of syllables. **Supervised Learning:** This type of learning requires a teacher that tells us what the right answers are and corrects us, if we get something wrong. For example, when teaching a kid the meaning of a word, we explicitly tell them what this word means, and if they mislabel something, e.g., call a small dog a cat, we correct them. **Reinforcement Learning:** This kind of learning-by-doing again happens naturally when humans learn from the consequences of their actions. For example, through experimentation and practice, we can figure out a complex sequence of hand movements to elicit beautiful sounds from a violin instead of producing painful screeches. While no single hand movement by itself is inherently good or bad, only the right combination will bring music to our ears.

Analogously, machines can also learn by following these three strategies:

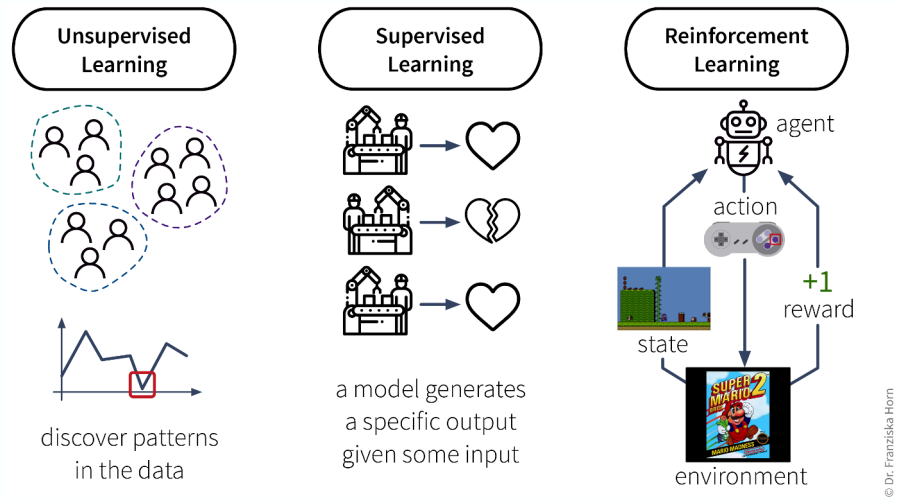
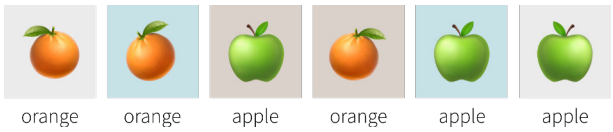


Figure 4: **Unsupervised Learning:** These algorithms pick up on statistical regularities in the data, for example, they can find groups of similar items (like in the customer segmentation task) or identify individual points that stand out (i.e., anomaly detection), e.g., unusual behavior of a machine due to a broken part or suspicious credit card transactions. **Supervised Learning:** These algorithms learn from many input-output examples, e.g., images and what is shown on these images or production conditions and whether the product that was produced under these conditions is faulty or okay. The learned model can then be used to predict the output for some new input. **Reinforcement Learning:** This type of learning is a bit more involved: Here the learning algorithm is also called an agent, which operates within an environment, e.g., a robot moving around in the real world or a virtual agent inside a simulation environment like a video game (which is usually much cheaper ;-)). The environment lets the agent know in which state or situation it currently is, then the agent can select how to react in this state, i.e., which (predefined) action to take, and then the environment determines the consequences of this action (e.g., kill a monster in a video game or fall off a cliff) and returns a reward depending on the outcome (e.g., extra points for collecting coins). Then the cycle repeats as the agent is in the next state. Based on the received reward, the agent learns over time which actions are beneficial in which situations and how to navigate the environment. The hard part here is that the reward signals often come much later after the action was executed, for example, in a video game, an agent collects a key at the beginning of a level, but the door that can be opened with this key comes many frames later, which means the reward will be delayed and the agent has a hard time associating this reward with the appropriate action. Since humans have a lot of background knowledge, figuring out what works and what doesn't in a game is much easier for us.

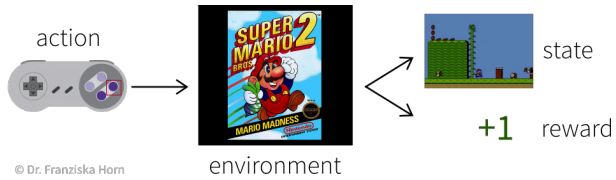
Data requirements for learning according to these strategies:

- Unsupervised Learning: a dataset with examples

- Supervised Learning: a dataset with ***labeled*** examples



- Reinforcement Learning: a (simulation) environment that generates data (i.e., reward + new state) in response to the agent's actions



With its reliance on a data-generating environment, reinforcement learning is a bit of a special case. Furthermore, as of now it's still really hard to get reinforcement learning algorithms to work correctly, which means they're currently mostly used in research and not so much for practical applications.

Supervised Learning

Supervised learning is the most common type of machine learning used in today's applications.

The goal here is to learn a **model (= a mathematical function)** $f(x)$ that describes the relationship between some **input(s)** x (e.g., different process conditions like temperature, type of material, etc.) and **output** y (e.g., resulting product quality).

This model can then be used to **make predictions for new data points**, i.e., compute $f(x') = y'$ for some new x' (e.g., predict for a new set of process conditions whether the produced product will be of high quality or if the process should be stopped to not waste resources).

Supervised Learning in a nutshell:

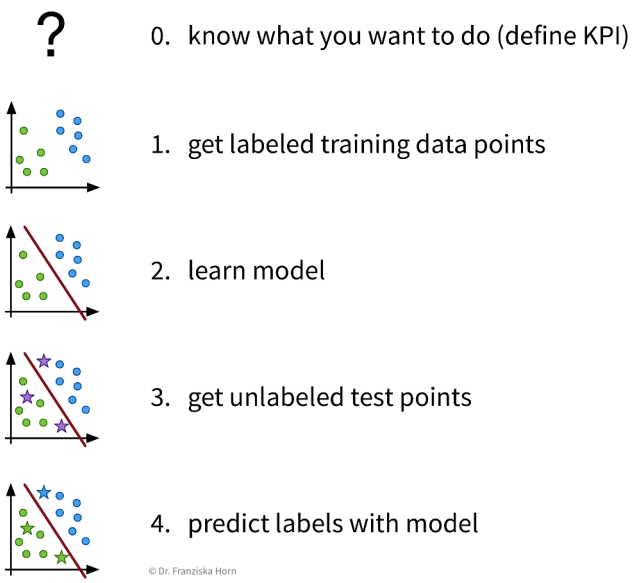


Figure 5: Before we start, we need to be very clear on what we want, i.e., what should be predicted, how will predicting this variable help us achieve our overall goals and create value, and how do we measure success, i.e., what is the Key Performance Indicator (KPI) of our process. Then, we need to collect data – and since we're using supervised learning, this needs to be *labeled* data, with the labels corresponding to the target variable that we want to predict. Next, we "learn" (or "train" or "fit") a model on this data and finally use it to generate predictions for new data points.

Features & Labels

A production process, where we want to predict whether a produced part is scrap given certain production conditions, is an example of a typical supervised learning problem. Here, the collected data for each produced part includes the process conditions under which it was produced, as well as the outcome, i.e., whether the product was okay or scrap:

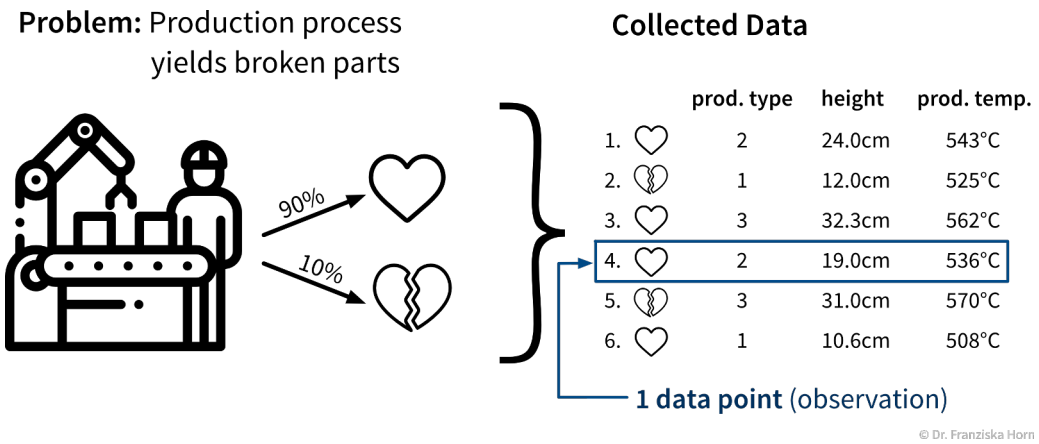


Figure 6: The data collected for this use case is structured data in a tabular form (e.g., in an excel sheet). One data point / sample / observation is always in one row of this table.

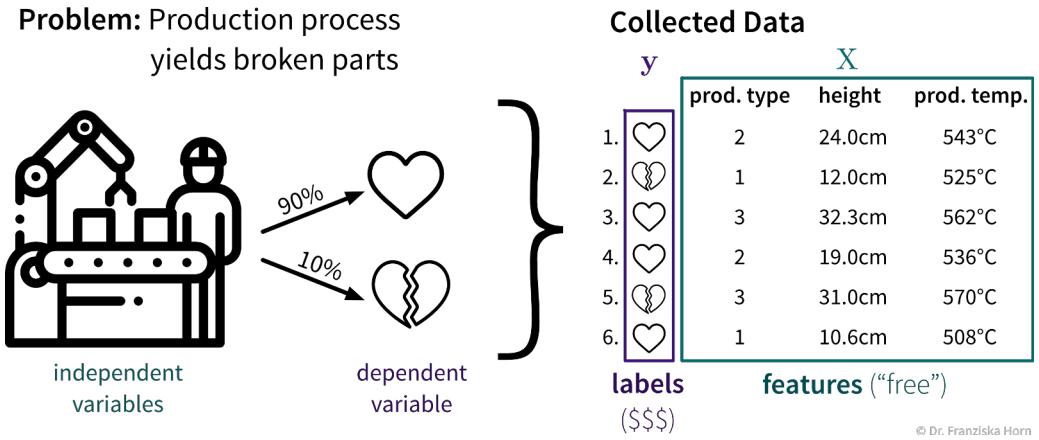
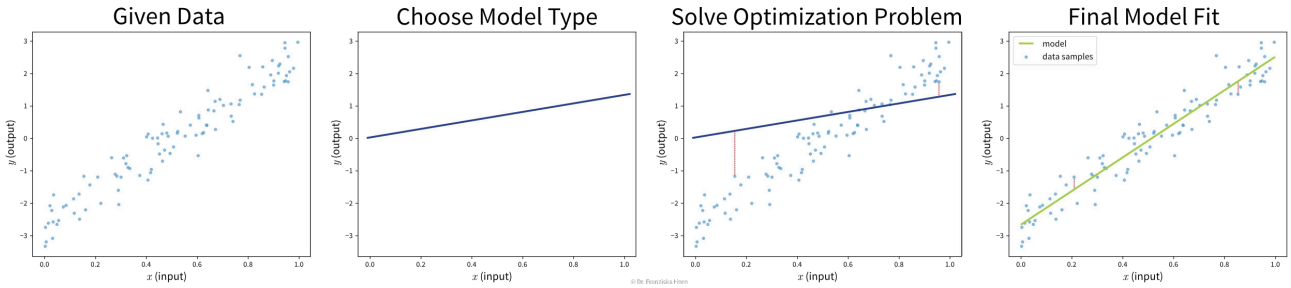


Figure 7: The columns of the table contain the different measurements / variables that were collected for each sample. Here we distinguish between *features* (in this case the production conditions) and *labels* (whether the product produced under these conditions is okay or scrap). Features, also denoted as a matrix X, are typically those measurements that we get basically for free, as they are often collected during the process for other purposes anyways. For example, when the operator of the machine sets the temperature for the production to a certain value, this is recorded as the signal is passed along to the heating unit. The corresponding labels, denoted as a vector $\mathbf{y}$, are often more expensive to collect. For example, in the production process, to collect a data point with the label *"scrap"*, we have to (intentionally) produce a broken product, costing us valuable resources. Another example: Google had to pay a team of specialist doctors to discuss and re-label some of the diabetic retinopathy images about which there existed conflicting opinions.

In the supervised learning setup, the features are used as the input to the model, while the labels constitute the target variable, i.e., the predicted output. Generally, features should be independent variables (e.g., settings that the operator can choose as he wishes), while the target value should be dependent on these inputs – otherwise we can't predict it from these inputs alone.

"Learning" a model from the data

Goal: Describe the relationship between input(s) x and output y with a model, i.e., a mathematical function $f(x)$



1. **Select a model class (= structure of the function):** Assumption: relationship is linear → linear regression model: $y = f(x) = b + w \cdot x$
2. **Define an objective:** Minimize error between true & predicted y: → $\min_{b,w} \sum_i (y_i - f(x_i))^2$
3. **Find best model parameters given the data:** i.e., solve the optimization problem defined in step 2
   $f(x) = -2.7 + 5.2x$

> **Video Recommendation**
>
> **If you're not familiar with linear regression**, the most basic supervised learning algorithm, please watch the explanation from Google decision scientist Cassie Kozyrkov on how linear regression works: [Part 1] [Part 2] [Part 3]

The available supervised learning algorithms differ in the **type of $x \rightarrow y$ relationship** they can describe (e.g., linear or nonlinear) and what kind of **objective** they minimize (also called loss function; an error computed on the training data, quantifying the mismatch between true and predicted labels). The task of a data scientist is to select a type of model that can optimally fit the given data. The rest is then taken care of by an **optimization method, which finds the parameters of the model that minimize the model's objective**, i.e., such that the model's prediction error on the given data is as small as possible.

> **Note**
>
> In most of the book, the terms "ML algorithm" and "ML model" will be used interchangeably. To be more precise, however, in general the algorithm processes the data and learns some parameter values. These parameter settings define the final model. For example, a linear regression *model* is defined by its coefficients (i.e., the model's parameters), which are found by executing the steps outlined in the linear regression *algorithm*, which includes solving an optimization problem.

Don't stop there!

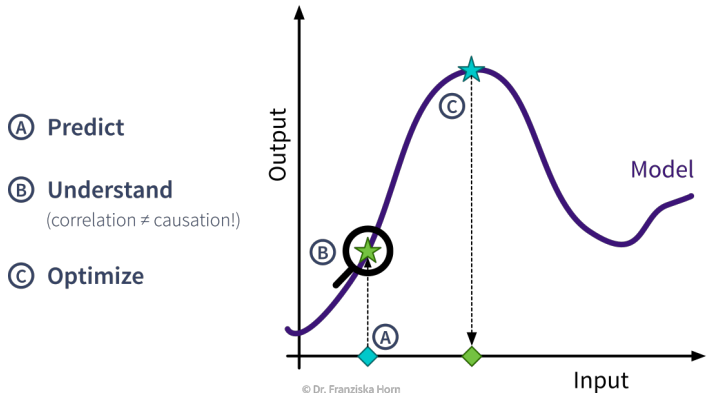


Figure 8: In many use cases, it is not enough to "only" predict the target for a new data point, e.g., predict whether a product produced under certain conditions will be of high quality or not. Instead, it is often necessary to additionally be able to explain *why* this prediction was made, e.g., which input feature values were the deciding factors, both to better understand possible root causes of a problem, but also to be assured that the model is basing its predictions on reasonable assumptions. Furthermore, a learned model can also be used within an outer optimization loop, i.e., in the simplest case one could systematically check what product quality the model predicts for different process conditions and then select the settings with the highest predicted quality to produce new products. But keep in mind that ML models are only built to interpolate, not extrapolate, i.e., make sure the settings that are tested are withing the training domain.

Predictive Analytics

By feeding historical data to a supervised learning algorithm, we can generate a **predictive model** that makes predictions about future scenarios to aid with planning.
Example: *Use sales forecasts to better plan inventory levels.*

Interpreting Predictive Models

Given a model that makes accurate predictions for new data points, we can interpret this model and explain its predictions to **understand root causes** in a process.
Example: *Given a model that predicts the quality of a product from the process conditions, identify which conditions result in lower quality products.*

What-if Analysis & Optimization

Given a model that makes accurate predictions for new data points, we can use this model in a **"what-if" forecast** to explore how a system might react to different conditions to make better decisions (but use with caution!).
Example: *Given a model that predicts the remaining lifetime of a machine component under some process conditions, simulate how quickly this component would deteriorate if we changed the process conditions.*

Going one step further, this model can also be used inside an optimization loop to automatically evaluate different inputs with the model systematically to **find optimal settings**.
Example: *Given a model that predicts the quality of a product from the process conditions, automatically determine the best production settings for a new type of raw material.*

ML use cases

The inputs that the ML algorithms operate on can come in many forms…

Structured vs. unstructured data

Data can come in various forms and while some data types require additional preprocessing steps, in principle ML algorithms can be used with all kinds of data.

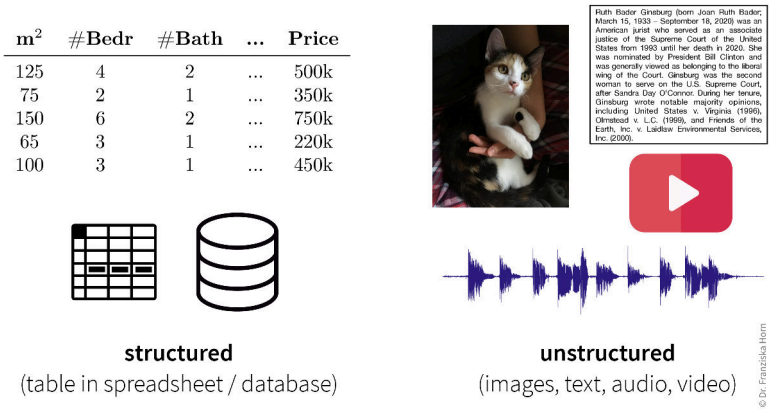


Figure 9: The main distinction when characterizing data is made between *structured* data, which is any dataset that contains individual measurements / variables / attributes / features that represent unique quantities, and *unstructured* data, which can not be subdivided into meaningful variables. For example, in images "first pixel from the left" or in texts "10th word in the second paragraph" is not what we would call a variable, while "size in square meters" and "number of bedrooms" are useful quantities to describe an apartment. Structured data is often *heterogeneous*, since the different variables in a dataset typically stand for very different things. For example, when working with sensor data, a dataset normally does not consist of only temperature measurements, but additionally it could contain, e.g., pressure and flow values, which have different units and measurement scales. Unstructured data, on the other hand, is *homogeneous*, e.g., there is no qualitative difference between the 10th and the 100th pixel in an image.

…but our goal, i.e., the desired outputs, determines the **type of algorithm** we should use for the task:

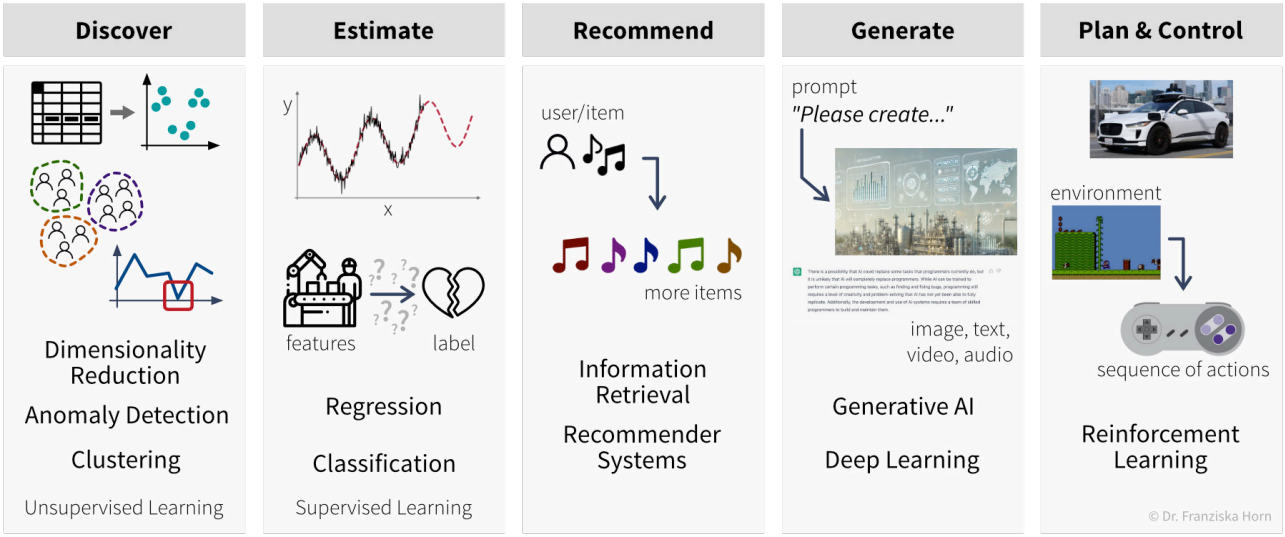


Figure 10: If our goal is to **discover** patterns in a dataset, unsupervised learning algorithms are ideal: *Dimensionality Reduction* provides an overview of the data by visualizing it in 2D, *Anomaly Detection* identifies outliers (e.g., a malfunctioning machine or a fraudulent credit card transaction), and *Clustering* groups similar samples (e.g., for customer segmentation). Supervised learning models are used to **estimate** unknown values from the given inputs (e.g., predict whether a product will be faulty if it is produced under certain conditions): *Regression* predicts continuous values (e.g., number of users, price, etc.), while *Classification* assigns discrete labels (e.g., an animal in a picture can either be a cat or a dog, but not something in between).
Recommender Systems and *Information Retrieval* algorithms can **recommend** items of interest, such as documents, songs, or movies, based on a user's preferences or items they've engaged with.
The most versatile are *Generative AI* and *Deep Learning* models, which primarily use unstructured data. They can **generate** diverse outputs—like images, text (e.g., for machine translation), or music—based on a given prompt.
Finally, *Reinforcement Learning* algorithms are used to **plan and control** processes by determining optimal action sequences under specific environmental conditions.

Some example 'input → output' tasks and what type of ML algorithm solves them:

Input X	Output Y	ML Algorithm Category
questionnaire answers	customer segmentation	clustering
sensor measurements	everything normal?	anomaly detection
past usage of a machine	remaining lifetime	regression
email	spam (yes/no)	classification (binary)
image	which animal?	classification (multi-class)
user's purchases	products to show	recommender systems
search query	relevant documents	information retrieval
audio	text	speech recognition
text in English	text in French	machine translation

To summarize (see also: overview table as PDF):

Existing ML solutions & corresponding output (for one data point):

- Dimensionality Reduction: (usually) **2D coordinates** (to create a visualization of the dataset)
- Outlier/Anomaly Detection: **anomaly score** (usually a value between 0 and 1 indicating how likely it is that this point is an outlier)
- Clustering: **cluster index** (a number between 0 and *k-1* indicating to which of the *k* clusters a data point belongs (or -1 for outliers))
- Regression: a **continuous value** (any kind of numeric quantity that should be predicted)
- Classification: a **discrete value** (one of several mutually exclusive categories)
- Generative AI: **unstructured output like a text or image** (e.g., speech recognition, machine translation, image generation, or neural style transfer)
- Recommender Systems & Information Retrieval: **ranking of a set of items** (recommender systems, for example, rank the products that a specific user might be most interested in; information retrieval systems rank other items based on their similarity to a given query item)
- Reinforcement Learning: a sequence of **actions** (specific to the state the agent is in)

Let's start with a more detailed look at the different unsupervised & supervised learning algorithms and what they are good for:

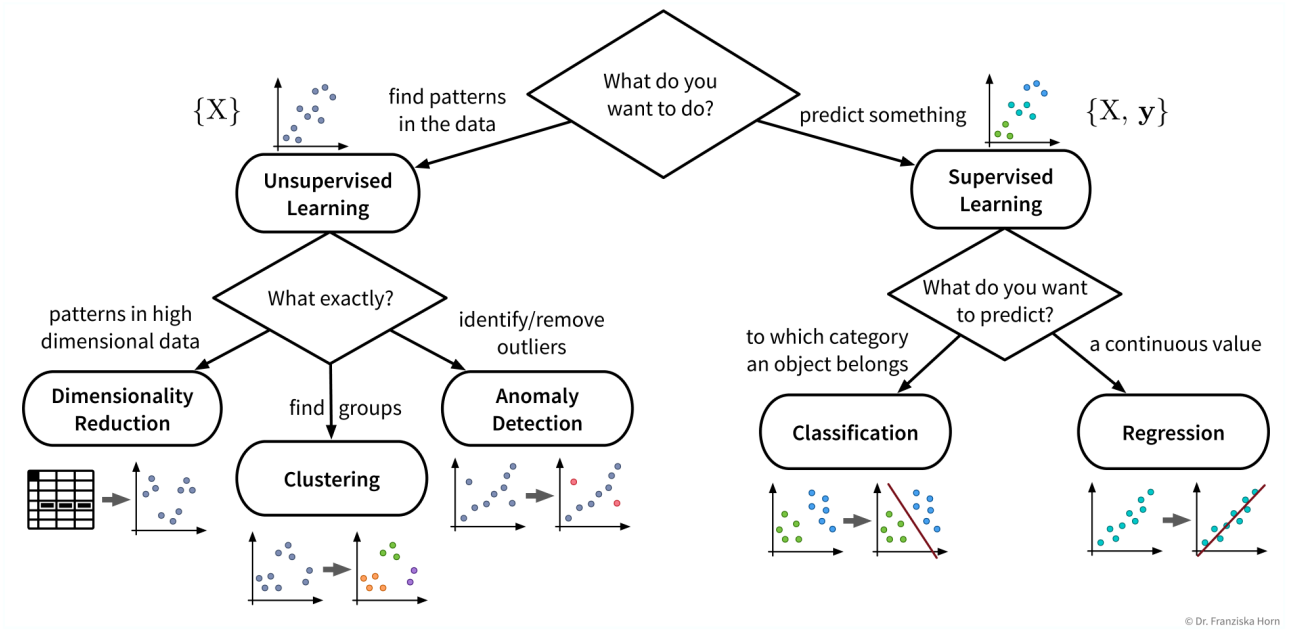


Figure 11: To apply unsupervised learning algorithms, we only need a feature matrix X, while learning a prediction model with supervised learning algorithms additionally requires the corresponding labels $\mathbf{y}$.

> **Tip**
>
> Even if our ultimate goal is to predict something (i.e., use supervised learning), it can still be helpful to first use unsupervised learning to get a better understanding of the dataset, for

example, by visualizing the data with dimensionality reduction methods to see all samples and their diversity at a glance, by identifying outliers to clean the dataset, or, for classification problems, by first clustering the samples to check whether the given class labels match the naturally occurring groups in the data or if, e.g., two very similar classes could be combined to simplify the problem.

Same dataset, different use cases

To illustrate the usefulness of the five different types of unsupervised and supervised learning algorithms, lets apply them to this example dataset:

m^2	# Bedr	# Bath	Renovated	...	Price	Sold
125	4	2	2000	...	500k	1
75	2	1	1990	...	350k	1
150	6	2	2010	...	750k	0
...	...	...	...	...	...	...
35	5	2	1999	...	620k	0
65	3	1	2015	...	220k	1
100	3	1	2003	...	450k	0

Table 2: This is a small toy dataset with structured data about different apartments, which someone might have gathered from a real estate website. It includes the size of the apartment in square meters, the number of bedrooms, the number of bathrooms, the year it was last renovated, and finally the price of the listing and whether it was sold for this price (1) or not (0).

Dimensionality Reduction

Use Cases:

- create a 2D visualization to explore the dataset as a whole, where we can often already visually identify patterns like samples that can be grouped together (clusters) or that don't belong (outliers)
- noise reduction and/or feature engineering as a data preprocessing step to improve the performance in the following prediction task

Example Unsupervised Learning: Dimensionality Reduction

Goal: Visualize the dataset

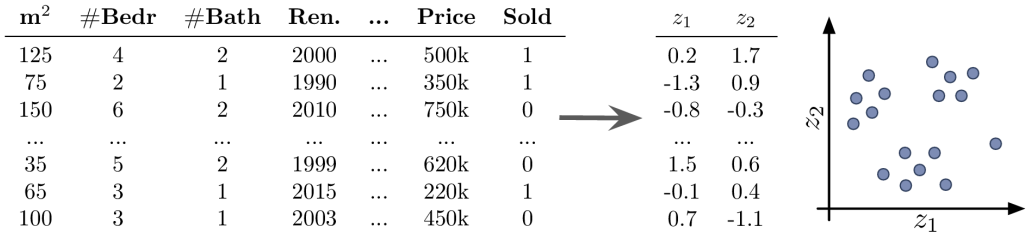


Figure 12: The first step when working with a new dataset is usually to visualize it, to get a better overview of all the samples and their diversity. This is done with a dimensionality reduction algorithm, which takes the original high dimensional data as input, where each column (= feature) in the table is one dimension, and outputs a lower dimensional representation of the samples, i.e., a new matrix with fewer columns (usually two for a visualization). With these two new features, here called z_1 and z_2, we can create a scatter plot of the dataset, where each sample / row (in this case each apartment) is represented as one point in this new 2D coordinate system. We can think of this plot as a map of our dataset that enables us to view all data points at a glance. This plot often shows interesting patterns, for example, groups of similar points, which would be located close to each other in this 2D map. Please note that for most dimensionality reduction methods, it is not possible to describe what is behind this new coordinate system. Specifically, these are not just the two most informative original features, but completely new dimensions that summarize the information of the original inputs. To better interpret these plots, it is helpful to color the dots afterwards by some variable, which can then reveal the driving factors behind the most salient patterns in the dataset. In this example, we could have used the price of each apartment to color the respective dot in the map, which might then reveal that similarly priced apartments are arranged next to each other.

Possible challenges:

- transforming the data with dimensionality reduction methods constructs new features as a (non)linear combination of the original features, which decreases the interpretability of the subsequent analysis results

Anomaly Detection

Use Cases:

- clean up the data, e.g., by removing samples with wrongly entered values, as a data preprocessing step to improve the performance in the following prediction task
- create alerts for anomalies, for example:
  - fraud detection: identify fraudulent credit card transaction in e-commerce
  - monitor a machine to see when something out of the ordinary happens or the machine might require maintenance

Example Unsupervised Learning: Anomaly Detection

Goal: Find outliers in the dataset

m^2	#Bedr	#Bath	Ren.	...	Price	Sold	Anomaly
125	4	2	2000	...	500k	1	0
75	2	1	1990	...	350k	1	0
150	6	2	2010	...	750k	0	0
...	...	...	...	...	...	...	...
35	5	2	1999	...	620k	0	1
65	3	1	2015	...	220k	1	0
100	3	1	2003	...	450k	0	0

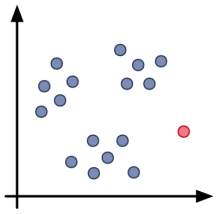


Figure 13: Next, we can check the dataset for outliers and then subsequently correct or remove these samples. An outlier detection algorithm outputs for each sample an anomaly score, which indicates whether this data point deviates from the norm. We can use these scores to colorize the 2D map of the dataset generated in the previous step to see the anomalies in context. One drawback is that an anomaly detection algorithm does not tell us *why* it considers an individual point an outlier. A data scientist needs to examine the points identified as outlier to see, e.g., if these should be removed due to flawed measurements or if they constitute some interesting edge cases. In this example, the sample identified as an anomaly is an apartment that supposedly has a size of only $35m^2$, but at the same time 5 bedrooms, i.e., most likely the person that originally entered the data made a mistake and the size of the listing should actually be $135m^2$.

Possible challenges:

- you should always have a good reason for throwing away data points – outliers are seldom random, sometimes they reveal interesting edge cases that should not be ignored

Clustering

Use Cases:

- identify groups of related data points, for example:
  - customer segmentation for targeted marketing campaign

Example Unsupervised Learning: Clustering

Goal: Find naturally occurring groups in the dataset

m^2	#Bedr	#Bath	Ren.	...	Price	Sold	Clustering
125	4	2	2000	...	500k	1	2
75	2	1	1990	...	350k	1	1
150	6	2	2010	...	750k	0	2
...	...	...	...	...	...	...	...
35	5	2	1999	...	620k	0	-
65	3	1	2015	...	220k	1	1
100	3	1	2003	...	450k	0	3

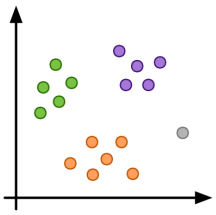


Figure 14: We can also check if the dataset contains naturally occurring groups. This is accomplished with a clustering algorithm, which returns a cluster index for each sample, where points with the same index are in the same cluster. Please note that these cluster indices are not ordered and when running the algorithm again, the samples might be assigned different numbers, however, the groups of samples that were assigned the same number should still be in a cluster together, i.e., this cluster might now just be called '5' instead of '3'. These cluster indices can again be used to colorize the 2D map of the dataset to see the clusters in context. While a clustering algorithm groups similar points together, it does not tell us *why* the points were assigned to a cluster and what this cluster means. Therefore, the data scientist again needs to examine the results to try to describe the different clusters. In our example, the clusters might be "cheap studio apartments", "large family apartments", and "luxurious penthouses". In unsupervised learning, there is no correct solution and a different algorithm might return different results. Just use the solution that is most helpful for your use case.

Possible challenges:

- no ground truth: difficult to choose between different models and parameter settings → the algorithms will always find something, but whether this is useful (i.e., what the identified patterns mean) can only be determined by a human in a post-processing step
- many of the algorithms rely on similarities or distances between data points, and it can be difficult to define an appropriate measure for this or know in advance which features should be compared (e.g., what makes two customers similar?)

Unsupervised learning has no ground truth

It is important to keep in mind that unsupervised learning problems have no right or wrong answers. Unsupervised learning algorithms simply recognize patterns in the data, which may or may not be meaningful for us humans.

For example, there exist a bunch of different unsupervised learning algorithms that group data points into clusters, each with a slightly different strategy and definition of what it means for two samples to be similar enough that they can be put into the same cluster.

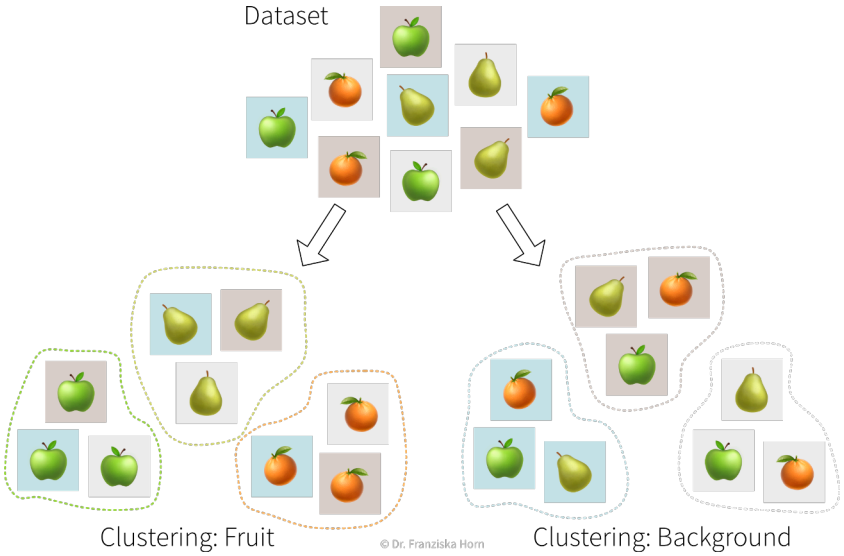


Figure 15: The first instinct of a human is to group these images according to the fruit displayed on them, however, there is nothing inherently wrong with clustering the images based on a different characteristic, such as their background color, whether or not the fruit has a leaf attached, in which direction the stem is pointing, etc.

It is up to the data scientist to examine the results of an unsupervised learning algorithm and make sense of them. And if they don't match our expectations, we can simply try a different algorithm.

Regression & Classification

Use Cases:

- Learn a model to describe an input-output relationship and make predictions for new data points, for example:
  - predict in advance whether a product produced under the proposed process conditions will be of high quality or would be a waste of resources
  - churn prediction: identify customers that are about to cancel their contract (or employees that are about to quit) so you can reach out to them and convince them to stay
  - price optimization: determine the optimal price for a product (often used for dynamic pricing, e.g., to adapt prices based on the device a customer uses (e.g., new iPhone vs old Android phone) when accessing a website)
  - predictive maintenance: predict how long a machine component will last
  - sales forecasts: predict revenue in the coming weeks and how much inventory will be required to satisfy the demand

Example Supervised Learning: Classification

Goal: Predict a discrete value for each data point

m^2	#Bedr	#Bath	Ren.	...	Price	Sold (predicted)	Sold	Error
125	4	2	2000	...	500k	1	1	0
75	2	1	1990	...	350k	0	1	1
150	6	2	2010	...	750k	0	0	0
...	...	...	...	...	...	...	...	...
35	5	2	1999	...	620k	0	0	0
65	3	1	2015	...	220k	1	1	0
100	3	1	2003	...	450k	1	0	1

Figure 16: Next, we can predict whether an apartment will be sold for the listed price. Since the variable "sold" only takes on the discrete values 'yes' (1) or 'no' (0), this is a binary classification problem. A classification model uses the attributes of an apartment together with the listing's price as inputs and predicts whether the apartment will be sold for this price. Since we have the true labels available for the initially collected dataset, we can evaluate how well the model performed by computing the number of wrong predictions it generated. This is the nice thing about supervised learning: We can objectively determine how good a solution is and benchmark different models against each other, while in unsupervised learning the data scientist needs to manually examine the results to make sense of them.

Example Supervised Learning: Regression

Goal: Predict a continuous value for each data point

m^2	#Bedr	#Bath	Ren.	...	Price (predicted)	Price	Error
125	4	2	2000	...	450k	500k	50k
75	2	1	1990	...	320k	350k	30k
150	6	2	2010	...	800k	750k	-50k
...	...	...	...	...	...	...	...
35	5	2	1999	...	560k	620k	60k
65	3	1	2015	...	250k	220k	-30k
100	3	1	2003	...	430k	450k	20k

Figure 17: Finally, we can predict a reasonable price for a listing. Since prices are continuous values, this is a regression problem, where the model uses as inputs the attributes of the apartments and predicts a suitable price. Again we have the true prices available and can compute the deviation of the regression model's estimates from the original price set by a real estate agent.

Possible challenges:

- success is uncertain: while it is fairly straightforward to apply the models, it is difficult to determine in advance whether there even exists any relation between the measured inputs and targets ($\rightarrow$ beware of garbage in, garbage out!)
- appropriate definition of the output/target/KPI that should be modeled, i.e., what does it actually mean for a process to run well and how might external factors influence this definition (e.g., can we expect the same performance on an exceptionally hot summer day?)

- missing important input variables, e.g., if there exist other influencing factors that we haven't considered or couldn't measure, which means not all of the target variable's variance can be explained
- lots of possibly irrelevant input variables that require careful feature selection to avoid spurious correlations, which would result in incorrect 'what-if' forecasts since the true causal relationship between the inputs and outputs isn't captured
- often very time intensive data preprocessing necessary, e.g., when combining data from different sources and engineering additional features

Deep Learning & Generative AI

Use Cases:

- automate tedious, repetitive tasks otherwise done by humans, for example (see also ML is everywhere!):
  - text classification (e.g., identify spam / hate speech / fake news; forward customer support request to the appropriate department)
  - sentiment analysis (subtask of text classification: identify if text is positive or negative, e.g., to monitor product reviews or what social media users are saying about your company)
  - speech recognition (e.g., transcribe dictated notes or add subtitles to videos)
  - machine translation (translate texts from one language into another)
  - image classification / object recognition (e.g., identify problematic content (like child pornography) or detect street signs and pedestrians in autonomous driving)
  - image captioning (generate text that describes what's shown in an image, e.g., to improve the online experience for for people with visual impairment)
  - predictive typing (e.g., suggest possible next words when typing on a smartphone)
  - data generation (e.g., generate new photos/images of specific objects or scenes)
  - style transfer (transform a given image into another style, e.g., make photos look like van Gogh paintings)
  - separate individual sources of an audio signal (e.g., unmix a song, i.e., separate vocals and instruments into individual tracks)
- replace classical simulation models with ML models: since exact simulation models are often slow, the estimation for new samples can be speed up by instead predicting the results with an ML model, for example:
  - AlphaFold: generate 3D protein structure from amino acid sequence (to facilitate drug development)
  - SchNet: predict energy and other properties of molecules given their configuration of atoms (to speed up materials research)

Possible challenges:

- selecting a suitable neural network architecture & getting it to work properly; especially when replacing traditional simulation models it is often necessary to develop a completely new type of neural network architecture specifically designed for this task and inputs / outputs, which requires a lot of ML & domain knowledge, intuition, and creativity
- computational resources (don't train a neural network without a GPU!)

- data quality and quantity: need a lot of *consistently* labeled data, i.e., many training instances labeled by human annotators who have to follow the same guidelines (but can be mitigated in some cases by pre-training the network using self-supervised learning)

Information Retrieval

Use Cases:

- improve search results by identifying similar items: given a query, rank results, for example:
  - return matching documents / websites given a search query
  - show similar movies given the movie a user is currently looking at (e.g., same genre, director, etc.)

Possible challenges:

- quality of results depends heavily on the chosen similarity metric; identifying semantically related items is currently more difficult for some data types (e.g., images) than others (e.g., text)

Recommender Systems

Use Cases:

- personalized suggestions: given a sample from one type of data (e.g., user, protein structure), identify the most relevant samples from another type of data (e.g., movie, drug composition), for example:
  - show a user movies that other users with a similar taste also liked
  - recommend molecule structures that could fit into a protein structure involved in a certain disease

Possible challenges:

- little / incomplete data, for example, different users might like the same item for different reasons and it is unclear whether, e.g., a user didn't watch a movie because he's not interested in it or because he just didn't notice it yet

Reinforcement Learning

Use Cases:

- Determine an optimal sequence of actions given changing environmental conditions, for example:
  - virtual agent playing a (video) game
  - robot with complex movement patterns, e.g., picking up differently shaped objects from a box

Unlike in regular optimization, where the optimal inputs given a single specific external condition are determined, here an "agent" (= the RL algorithm) tries to learn an optimal *sequence* of inputs to maximize the cumulative reward received over multiple time steps, where there can be a significant time delay between the inputs and the rewards that they generate (e.g., in a video game we might need to pick up a key in the beginning of a level, but the door that can be opened with it only comes several frames later).

Possible challenges:

- usually requires a simulation environment for the agent to learn in before it starts acting in the real world, but developing an accurate simulation model isn't easy and the agent will exploit any bugs if that results in higher rewards
- can be tricky to define a clear reward function that should be optimized (imitation learning is often a better option, where the agent instead tries to mimic the decisions made by a human in some situation)
- difficult to learn correct associations when there are long delays between critical actions and the received rewards
- agent generates its own data: if it starts off with a bad policy, it will be tricky to escape from this (e.g., in a video game, if the agent always falls down a gap instead of jumping over it, it never sees the rewards that await on the other side and therefore can't learn that it would be beneficial to jump over the gap)

Other

> **!** Important
>
> ML algorithms are categorized by the output they generate for each input. If you want to solve an 'input → output' problem with a different output than the ones listed above, you'll likely have to settle in for a multi-year research project – if the problem can be solved with ML at all!

To solve complex problems, we might need multiple algorithms

Example: virtual assistant (e.g., Siri or Alexa): *"Hey <smart speaker>, tell me a joke!"* → a random joke

This might look like an input-output problem, but it would be very difficult and inefficient to solve it directly. Instead, we break the problem down into smaller subtasks that can be solved with existing algorithms:

1. **Trigger word detection:**
   audio → "Hey <smart speaker>" (yes/no)?
2. **Speech recognition:**
   audio → text
3. **Intent classification:**
   text → (joke/timer/weather/…)?
4. **Request-specific program (e.g., select random joke)**

5. **Speech generation:**
   text → audio

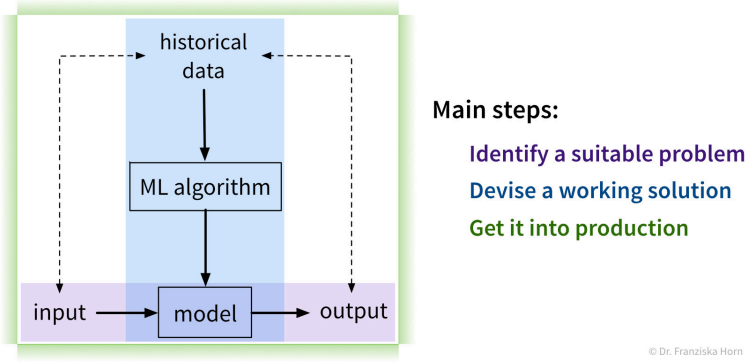
First, the smart speaker needs to know whether it was activated with a specific trigger word (e.g., "Hey Siri"). This is a simple binary classification task (trigger word: yes/no), which is usually performed on the device itself, since we don't want that everything we say is continuously streamed into the cloud. Next, the spoken words that follow the trigger word are transcribed into text. Text is easier to handle, because, for example, variations due to different accents are removed. Based on this text, the intent is recognized, i.e., which of the different functionalities of the virtual assistant should be used (e.g., tell a joke, play music, set an alarm, etc.). This is a multi-class classification problem. The next step is to execute the request, which is not done with ML, but instead some task-specific program is run, e.g., to select a joke from a database or set a timer, etc., based on the apps installed on the device. Finally, the output of the program needs to be converted back into an audio signal. For this again an ML model can help to get smoothly spoken text – and in the near future maybe with the voice of Morgan Freeman or some other famous person like in "Deep Fake" applications.

It is generally advisable to first think about how a problem could be decomposed into easier-to-solve subproblems, especially since there might already be a large dataset or pre-trained ML model available for one of these subtasks. For example, speech recognition models can be trained on audio books and transcribed political speeches in addition to the data collected from the smart speaker users.

> **Caution**
>
> When one ML model receives as input the output of another ML model, this means as soon as we roll out a new version of the ML model at the beginning of the chain, we should also retrain the models following this one, since they might now receive slightly different inputs, i.e., experience a data drift.

Solving problems with ML

Solving "input → output" problems with ML requires three main steps:

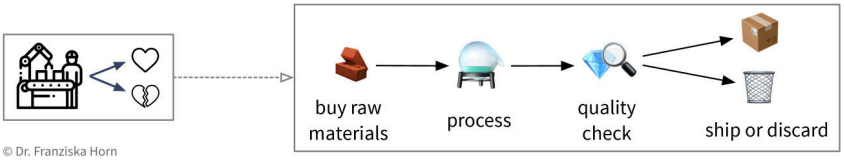


1. Identify a suitable problem

The first (and arguably most important) step is to **identify where machine learning can (and should) be used** in the first place.

Steps to identify a potential ML project

1. Create a process map: which steps are executed in the business process (flow of materials & information) and what data is collected where. For example, in a production process where some of the produced parts are defective:



2. Identify parts of the process that could either be automated with ML (e.g., straightforward, repetitive tasks otherwise done by humans) or in other ways improved by analyzing data (e.g., to understand root causes of a problem, to improve planning with what-if simulations, or to optimize the use of resources):

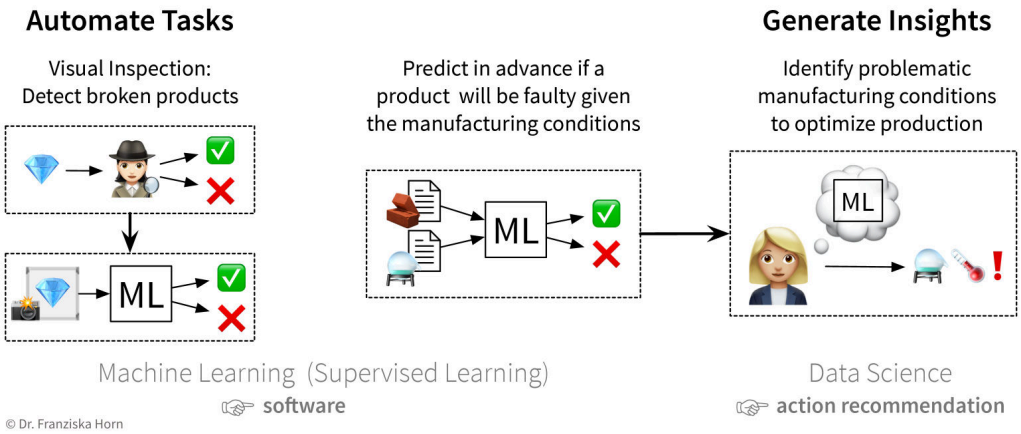


Figure 18: The first idea is to automate the quality check that was so far done by a human: since the human can easily recognize the defects in the pictures taken of the products, an ML model should be able to do this, too. The next idea is to try to predict in advance whether a product will be faulty or not based on the composition of raw materials and the proposed process conditions: success here is unclear, since the human experts are not sure whether all of the information necessary to determine if the product will be fine is contained in this data – but nevertheless it's worth a try since this could save lots of resources. While the final ML model that solves the input-output problem can be deployed as software in the ongoing process, when a data scientist analyzes the results and interprets the model, she can additionally generate insights that can be translated into action recommendations.

3. Prioritize: which project will have a high impact, but at the same time also a good chance of success, i.e., should yield a high return on investment (ROI)? For example, using ML to automate a simple task is a comparatively low risk investment, but might cause some assembly-line workers to loose their jobs. In contrast, identifying the root causes of why a production process results in 10% scrap could save millions, but it is not clear from the start that such an analysis will yield useful results, since the collected data on the process conditions might not contain all the needed information.

ML project checklist

Motivation

- **What problem do you want to solve?**
  Machine learning can help you in various ways by generating insights from large amounts of (possibly unstructured) data, improving decision making and planning processes by providing predictions about future events, or automating tedious tasks otherwise requiring human experts. Where do you see a lot of inefficiencies around you that could be mitigated by a better use of data? For example, you could look for opportunities to decrease wasted resources / time / costs or increase revenue / customer satisfaction / etc.
  To systematically identify problems or opportunities, it can be helpful to create a process map or customer journey map.
- **In what way(s) would this generate value for your organization?**
  How could your organization make money on this or reduce costs?

  - Could this *improve an internal process* (e.g., maybe a process can be run more efficiently with the insights from an analysis or a tedious task that would otherwise require a human worker can be automated using an ML model)?
  - Could the ML model be integrated as a *new feature within an existing product* and thereby, e.g., make this product more appealing to customers?
  - Could the ML solution be sold as an entirely *new product or service*, e.g., offered as a *Software-as-a-Service (SaaS) solution*?

Please note that how the ML solution will be used in the end might also be a strategic decision that can be different for every organization. For example, an ML solution that recognizes scratches in produced products might be used by one company to improve their internal production process, while another company that produces the machines that make the products could integrate this as a new feature in their machines, and a third company might offer this as a SaaS solution compatible with different production lines.

- **How much value could this project generate?**
  Think of the impact in terms of

  - *Magnitude:* Small improvement or revolution? Will the solution result in a strategic advantage?
  - *Scale:* How often will this be used? How many users/customers/employees will benefit? For example:

    - Small process optimization, *but* since this process is used everyday in the whole organization it saves countless hours
    - New feature that revolutionizes the product and sets you apart from the competition, *but* the market for it is tiny
  - Would this have any *valuable side effects*? What will be different? Any additional opportunities that could arise from this? Can you create synergies between departments that work with similar data?

- **How do you know you've accomplished your goal?**
  What would success look like, i.e., what's your definition of 'done'?
  - Can you quantify the progress towards your goal with a KPI?
  - What is the status quo, i.e., how far are you from your goal right now? What is your target?
  - Which metrics should *not* change (i.e., get worse) due to this project?

Solution Outline

- **What is your vision for the future with ML?**
  - What does your existing process / system look like and how will it be different after you integrate the ML solution?
  - Who are the users and how will they be affected by this change, e.g., will they require additional training to use the new system?

- **What are the deliverables?**
  Does the solution consist of a piece of **software** that is deployed somewhere to continuously make predictions for new data points, or are you more interested in the **insights** gained from an one-off analysis of historical data?

- **In case of a software solution, how will the ML model be integrated with the existing setup?**
  - What does one interaction with the system look like (= 1 data point / sample / observation), e.g., a user making a request or a produced product passing a quality checkpoint?
  - Where are the inputs for the ML model coming from? What happens to the outputs of the ML model?
  - Do you need an additional user interface (UI) or API to interact with the ML model?
  - Does the ML model need to make predictions instantly as new data comes in or can it process data asynchronously in batches? What is the expected traffic (i.e., number of data points that need to be processed per second)?
  - How should the ML model be deployed (e.g., cloud, on-premise, or edge device)? Does this require any additional infrastructure or special hardware (e.g., GPUs)?
  - Model maintenance: What are the plans w.r.t. pipelines for future data collection, model monitoring, and automated retraining?

- **What is the input data? What should the outputs look like?**
  - What kind of inputs does the ML model receive (e.g., image / text / sensor measurements / etc.)?

  - What kind of outputs should the ML model produce, i.e., which category of ML algorithms solves this kind of problem?
  - Do you already have access to an initial dataset to train the model?

- **How will you evaluate the performance of the ML model?**

  - What evaluation metric is appropriate for the type of ML use case (e.g., accuracy)?
  - How does this evaluation metric relate to the business KPI this solution is supposed to improve?
  - How can the performance of the model be monitored during operation? Is new labeled data continuously collected for this purpose?

- **Is there a simpler solution, i.e., without using ML?**
  Use ML to learn *unknown, complex* rules from data.

  - Even if ML is the right choice here, could you build a minimal viable product without ML to already validate the solution as a whole before investing in ML?

Challenges & Risks

- **Is there enough high-quality data available to train and evaluate the model?**

  - Quality: Do you have the right inputs and unambiguous labels?
    → Ask a subject matter expert whether she thinks all the relevant input data is available to compute the desired output. This is usually easy to determine for unstructured data such as images – if a human can see the object in the image, ML should too. But for structured data, such as a spreadsheet with hundreds of columns of sensor measurements, this might be impossible to tell before doing any analysis on the data.
  - Quantity: How much data was already collected (including rare events and labels)? How long would it take to collect more data? Could additional data be bought from a vendor and if yes, how much would this cost?
  - How difficult is it to get access to all of the data and combine it neatly in one place? Who would you talk to, to set up / improve the data infrastructure?
  - How much preprocessing is necessary (e.g., outlier removal, fixing missing values, feature engineering, i.e., computing new variables from the existing measurements, etc.)? What should be the next steps to systematically improve data quality and quantity and decrease preprocessing requirements in the future?

- **Can the problem be solved with an existing ML algorithm?**
  Ask an ML expert whether a similar problem has already been solved before.

  - For known solutions: How complex is it to get the model working (e.g., linear regression vs. deep neural network)?
  - For unknown solutions: Instead of spending years on research to come up with a novel algorithm, is it possible to break the input-output problem down into simpler subproblems with known solutions?

- **What would be the worst case scenario when the model is wrong?**
  Your ML system (like humans) will make mistakes. Do not use ML if you always need 100% correct results!

  - What level of performance do you need at least for the ML solution to be valuable? E.g., what false positive or false negative rates are you willing to tolerate? Is the desired performance realistic with the given data? What would be the worst case scenario when the model produces wrong predictions and how much risk are you willing to take?
  - What is the chance of the input data changing over time, e.g., because of changing user demographics or black swan events like a pandemic (e.g., COVID-19)? How often would you need to retrain the model to compensate for these drifts and do you collect new (labeled) data quickly enough to do this?
  - Do users have an incentive to intentionally deceive the system (e.g., spammers who come up with more sophisticated messages if their original ones are caught by the spam filter; adversarial attacks)?
  - Instead of going all in with ML from day 1, is there a way your system can be monitored in the beginning while still providing added value (i.e., human-in-the-loop solution)?

- **Are there any potential legal issues or ethical concerns?**

  - Is the use of ML prohibited for this kind of application by some regulation, e.g., the EU AI Act?
  - Are there any concerns w.r.t. data privacy, e.g., because you are relying on personally identifiable information (PII)?
  - Do the decisions of the ML model need to be transparent and explainable, e.g., if someone is denied credit because of an algorithmically generated credit score?
  - Is there a risk of model discrimination, e.g., because the model is potentially trained on systematically biased data?

- **What else could go wrong?**

  - Why might users get frustrated with the solution? For example, when might they prefer to interact with a real human instead of a chatbot?

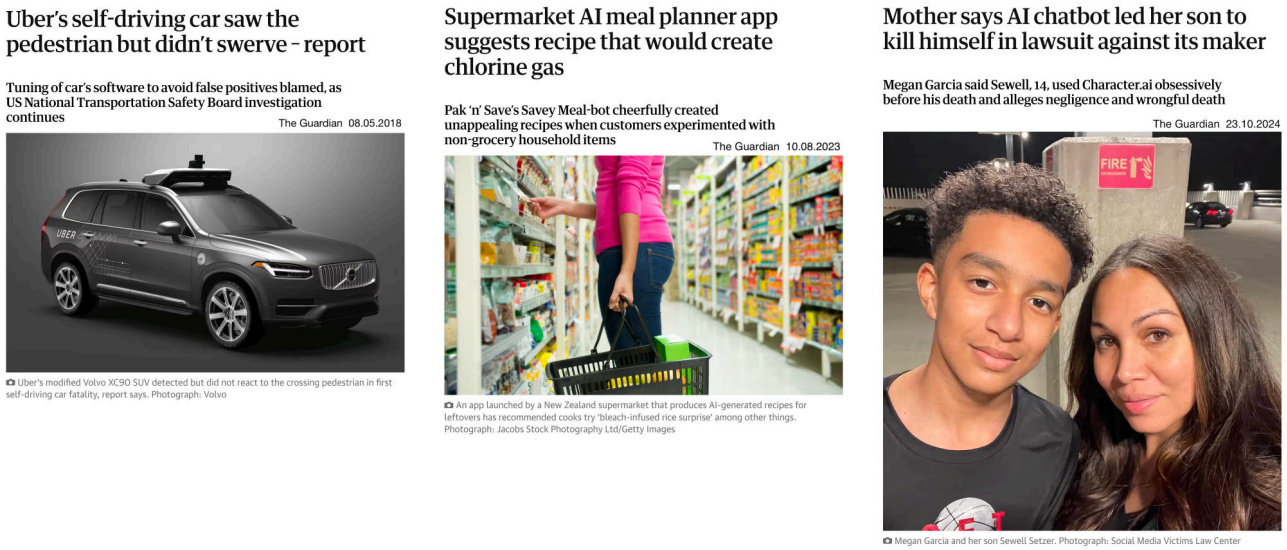


Figure 19: Fortunately, life-or-death situations are not a concern for most machine learning use cases. However, it is important to consider the worst-case scenario when the model produces incorrect outputs. Creating effective guardrails to prevent misuse is particularly difficult for more complex models like the large language models (LLMs) powering generative AI.

Build or Buy?

- **Core vs. generic domain: Does this create a strategic advantage?**
  Will the solution be a key part of your business, e.g., a new feature that makes your product more attractive, and/or does it require unique subject matter expertise only available at your organization, e.g., because you're analyzing data generated by your own specific processes/machines? Or is this a common (but complex) problem, for which a solution already exists (e.g., offered as a Software-as-a-Service (SaaS) product), that you could buy off the shelf?
  For example, extracting the relevant information from scanned invoices to automate bookkeeping processes is a relatively complex task for which many good solutions already exist, so unless you are working in a company building bookkeeping software and plan to sell a better alternative to these existing solutions, it probably doesn't make sense to implement this yourself.

- **Do you have the required technical and domain know-how to build this yourself?**

  - How difficult would it be to implement the ML solution yourself? For example, what kind of open source libraries already exist that could be used to solve such a task?
  - Do you have the necessary ML talent? If not, you could also consider a hybrid approach where you partner with an academic institution or external consultants.

- **What is the return on investment (ROI) for an off-the-shelf solution?**

  - How reliable is the off-the-shelf ML solution? Are there any benchmarks available and/or can you test it with some common examples and edge cases yourself?
  - How much effort would be required in terms of preprocessing your data before you could use the off-the-shelf ML solution?

- How difficult would it be to integrate the output from the off-the-shelf ML solution into your general workflow? Does it do exactly what you need or would additional post-processing steps be required?
- Can the off-the-shelf ML solution be deployed in-house or does it run on an external server and would this bring with it any data privacy issues?
- How high are the on-going licensing fees and what is included in terms of maintenance (e.g., how frequently are the models retrained)?

Unless the ML solution will be an integral part of your business, in the end it will probably come down to comparing costs for developing, implementing, running, and maintaining the system yourself vs. costs for integrating the off-the-shelf solution into your existing workflow (incl. necessary data preprocessing) and on-going licensing fees.

But even if you decide to build your own ML solution, you rarely start from scratch—you typically **build upon generic components**, such as open source libraries or pretrained models.

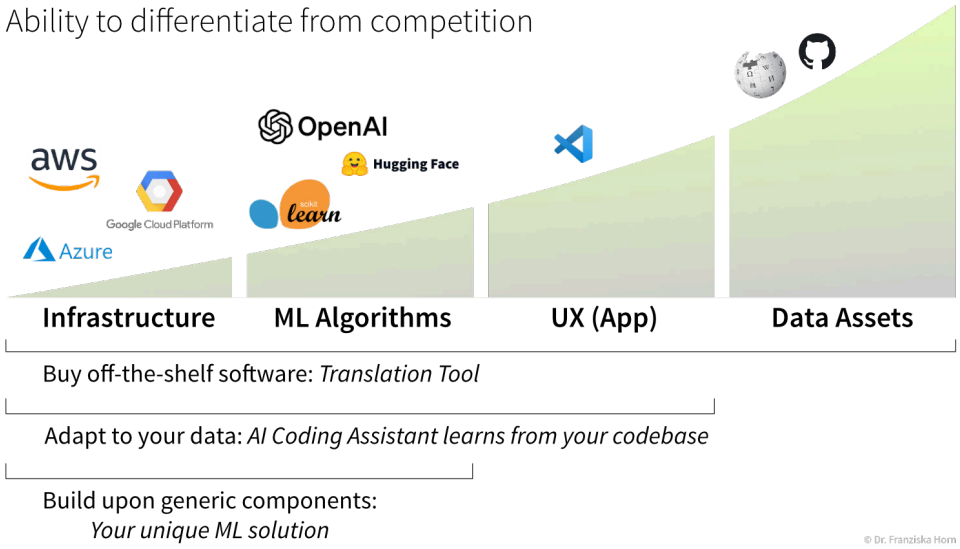


Figure 20: The "build or buy" decision is often made on a continuum: You can buy an off-the-shelf tool that works out-of-the-box; the software that you buy might get better if you finetune it on our own data; or you can build your own software, often by reusing generic components such as cloud infrastructure offerings or open source libraries and pretrained models.

In this context, it is also important to think about which parts of your ML-based product will be the most **difficult for competitors to replicate**: this is usually the **proprietary data** your models were trained on. While datasets can sometimes be scraped from the web (though this is highly controversial when it comes to copyright-protected materials), using your own data ensures a competitive edge—because it reflects the specific context of your business, cannot easily be copied, and often leads to better model performance.

For more details check out this blog article.

2. Devise a working solution

Once a suitable "input → output" problem as been identified, **historical data needs to be gathered and the right ML algorithm needs to be selected and applied** to obtain a working solution. This is what the next chapters are all about.

To solve a concrete problem using ML, we follow a workflow like this:

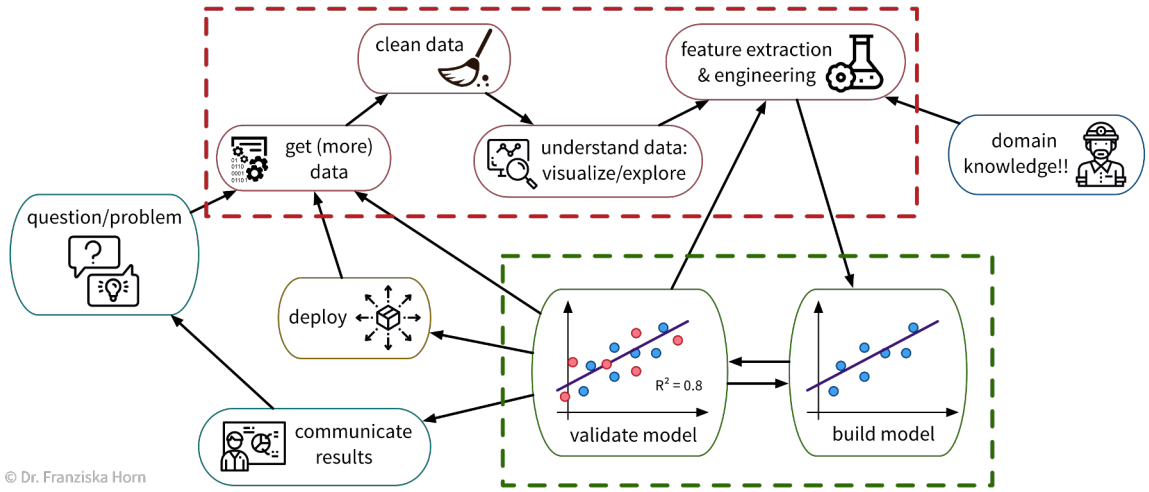
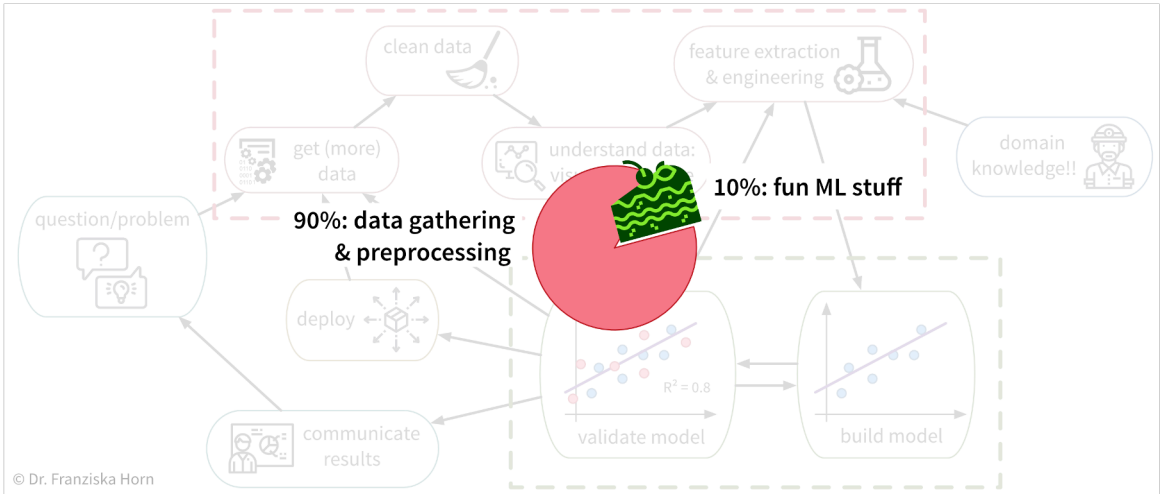


Figure 21: We always start with some kind of question or problem that should be solved with ML. And to solve it, we need data, which we most likely have to clean before we can work with it (e.g., merge different excel files, fix missing values, etc.). Then it's time for an exploratory analysis to better understand what we're dealing with. Depending on the type of data, we also need to extract appropriate features or engineer additional ones, for which domain knowledge / subject matter expertise is invaluable. All these steps are grouped under "preprocessing" *(red box)* and the steps are not linear, as we often find ourselves jumping back and forth between them. For example, by visualizing the dataset, we realize that the data contains some outliers that need to be removed, or after engineering new features, we go back and visualize the dataset again. Next comes the ML part *(green box)*: we normally start with some simple model, evaluate it, try a more complex model, experiment with different hyperparameters, … and at some point realize, that we've exhausted our ML toolbox and are still not happy with the performance. This means we need to go back and either engineer better features or, if this also doesn't help, collect more and/or better data (e.g., more samples, data from additional sensors, cleaner labels, etc.). Finally, when we're confident in the model's predictions, there are two routes we can take: Either the data science route, where we communicate our findings to the stakeholders (which most likely results in further questions). Or the ML software route, where the final model is deployed in production. Here it is important to continuously monitor the model's performance and collect new data such that the model can be retrained, especially as the inevitable data or concept drifts occur. Above all, working on a machine learning project is a very iterative process.

Unfortunately, due to a lack of standardized data infrastructure in many companies, the sad truth is that usually (at least) about 90% of a Data Scientist's time is spent collecting, cleaning, and otherwise

preprocessing the data to get it into a format where the ML algorithms can be applied:



While sometimes frustrating, the time spent cleaning and preprocessing the data is never wasted, as only with a solid data foundation the ML algorithms can achieve decent results.

3. Get it into production

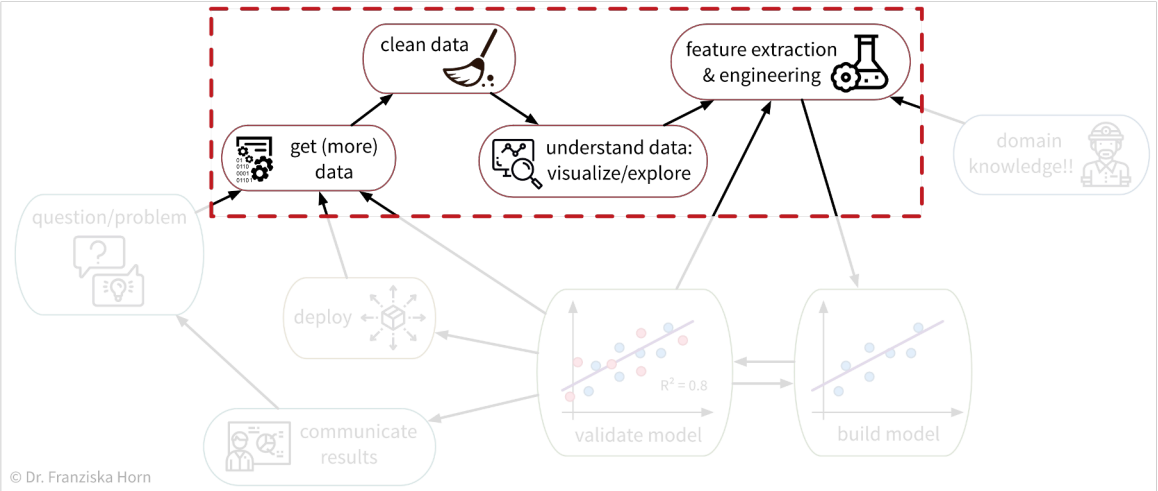
When the prototypical solution has been implemented and meets the required performance level, this solution then has to be deployed, i.e., **integrated into the general workflow and infrastructure** so that it can actually be used to improve the respective process in practice (as a piece of software that continuously makes predictions for new data points). This might also require building some additional software around the ML model such as an API to programmatically query the model or a dedicated user interface to interact with the system. Finally, there are generally two strategies for how to run the finished solution:

1. **The ML model runs on an "edge" device**, i.e., on each individual machine (e.g., mobile phone) where the respective data is generated and the output of the model is used in subsequent process steps. This is often the best strategy when results need to be computed in real time and / or a continuous Internet connection can not be guaranteed, e.g., in self-driving cars. However, the downside of this is that, depending on the type of ML model, comparatively expensive computing equipment needs to be installed in each machine, e.g., GPUs for neural network models.
2. **The ML model runs in the "cloud"**, i.e., on a central server (either on-premise or provisioned from a cloud provider such as AWS), e.g., in the form of a web application that receives data from individual users, processes it, and sends back the results. This is often the more efficient solution, if a response within a few seconds is sufficient for the use case. However, processing personal information in the cloud also raises privacy concerns. One of the major benefits of this solution is that it is easier to update the ML model, for example, when more historical data becomes available or if the process changes and the model now has to deal with slightly different inputs (we'll discuss this further in later chapters).

→ As these decisions heavily depend on your specific use case, they go beyond the scope of this book. Search online for "MLOps" or read the book Designing Machine Learning Systems to find out more about these topics and hire a machine learning or data engineer to set up the required infrastructure in your company.

Data Analysis & Preprocessing

As we've seen, ML algorithms solve input-output tasks. And to solve an ML problem, we first need to collect data, understand it, and then transform ("preprocess") it in such a way that ML algorithms can be applied:



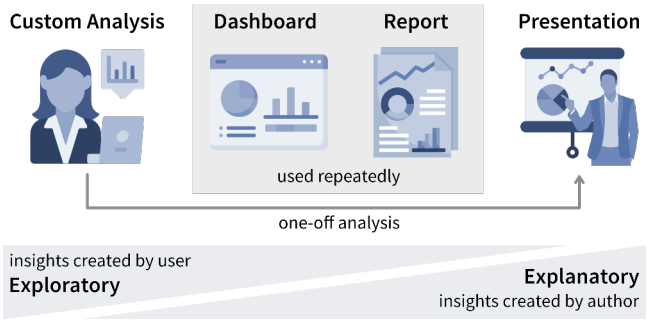
Data Analysis

Analyzing data is not only an important step before using this data for a machine learning project, but can also generate valuable insights that result in better (data-driven) decisions. We usually analyze data for one of two reasons:

1. We need some specific information to make a (better) decision (*reactive analysis*, e.g., when something went wrong and we don't know why).
2. We're curious about the data and don't know yet what the analysis will bring (*proactive analysis*, e.g., to better understand the data at the beginning of an ML project).

Data analysis results can be obtained and communicated in different formats

- A **custom analysis** with results presented, e.g., in a power point **presentation**
- A **standardized report**, e.g., in form of a PDF document, showing static data visualizations of historical data
- A **dashboard**, i.e., a web app showing (near) real-time data, usually with some interactive elements (e.g., options to filter the data)



While the data story that is told in a presentation is usually fixed, users have more opportunities to interpret the data and analyze it for themselves in an interactive dashboard.

What all forms of data analyses have in common is that we're after "(actionable) insights".

What is an insight?

Psychologist Gary Klein defines an insight as "an unexpected shift in the way we understand things".

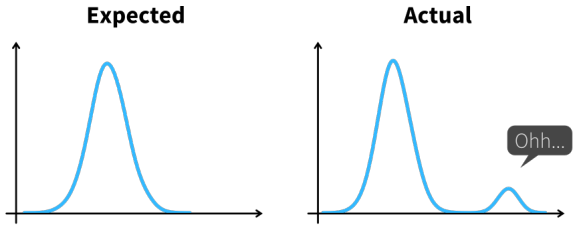


Figure 1: Things get interesting when we find something in the data that we didn't expect. [Adapted from: *Effective Data Storytelling* by Brent Dykes]

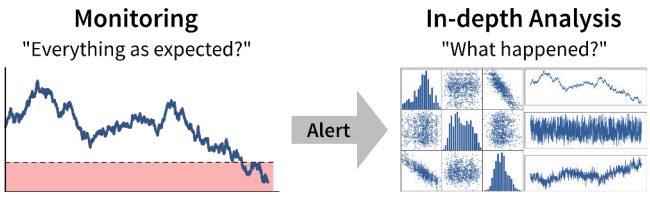
Arriving at an insight requires two steps:

1. **Notice something unexpected**, e.g., a sudden drop or increase in some metric.
2. **Understand why this happened**, i.e., dig deeper into the data to identify the root cause.

When we understand why something happened, we can often also **identify a potential action that could get us back on track**, thereby making this an *actionable insight*.

> **Tip**
>
> Knowing which values are unexpected and where it might pay off to dig deeper often requires some domain knowledge, so you might want to examine the results **together with a subject matter expert**.

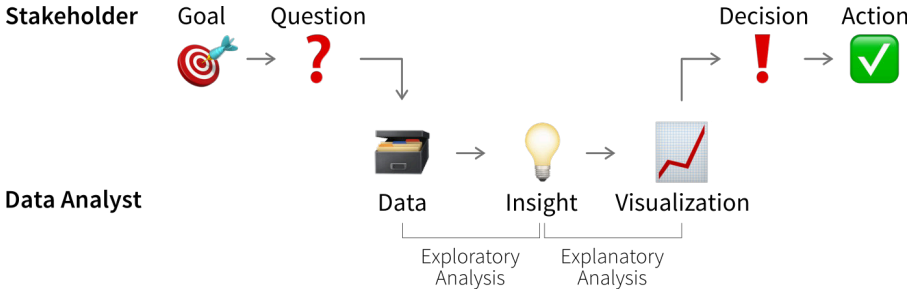
Ideally, we should continuously **monitor important metrics in dashboards or reports** to spot deviations from the norm as quickly as possible, while **identifying the root cause often requires a custom analysis**.



> **Tip**
>
> As a data analyst you are sometimes approached with **more specific questions or requests** such as "We're deciding where to launch a new marketing campaign. Can you show me the number of users for all European countries?". In these cases it can be helpful to **ask "why?" to understand where the person noticed something unexpected** that prompted this analysis request. If the answer is "Oh, we just have some marketing budget left over and need to spend the money somewhere" then just give them the results. But if the answer is "Our revenue for this quarter was lower than expected" it might be worth **exploring other possible root causes** for this, as maybe the problem is not the number of users that visit the website, but that many users drop out before they reach the checkout page and the money might be better invested in a usability study to understand why users don't complete the sale.

Data-driven Decisions

While learning something about the data and its context is often interesting and can feel rewarding by itself, it is not yet valuable. Insights become valuable when they influence a decision and inspire a different course of action, better than the default that would have been taken without the analysis.



This means we need to understand **which decision(s)** the insights from our data analysis should influence.

> **Note**
>
> Not all decisions need to be made in a data-driven way. But decision makers should be transparent and upfront about whether a decision can be influenced by analysis results, i.e., what data would make them change their mind and choose a different course of action. If data is only requested to support a decision that in reality has already been made, save the analysts the time and effort!

Before we conduct a data analysis we need to be clear on:

- Who are the **relevant stakeholders**, i.e., who will consume the data analysis results (= our audience / dashboard users)?
- What is **their goal**?

In business contexts, the users' goals are usually in some way related to making a profit for the company, i.e., increasing revenue (e.g., by solving a customer problem more effectively than the competition) or reducing costs.

The progress towards these goals is tracked with so called ***Key Performance Indicators* (KPIs)**, i.e., custom metrics that tell us how well things are going. For example, if we're working on a web application, one KPI we might want to track could be "user happiness". Unfortunately, true user happiness is difficult to measure, but we can instead check the number of users returning to our site and how long they stay and then somehow combine these and other measurements into a proxy variable that we then call "user happiness".

> **Caution**
>
> A KPI is only a reliable measure, if it is not simultaneously used to control people's behavior, as they will otherwise try to game the system (Goodhart's Law). For example, if our goal is high quality software, counting the number of bugs in our software is not a reliable measure for quality, if we simultaneously reward programmers for every bug they find and fix.

Lagging vs. Leading KPIs

Unfortunately, the things we really care about can often only be **measured after the fact, i.e., when it is too late to take corrective action**. For example, in sales we care about "realized revenue", i.e., money in the bank, but if the revenue at the end of the quarter is less than what we hoped for, we can only try to do better next quarter. These metrics are called **lagging KPIs**.
Leading KPIs instead tell us when we should **act before it's too late**. For example, in sales this could be the volume of deals in the sales pipeline, i.e., while not all of these deals might eventually go through and result in realized revenue, if the pipeline is empty we know for certain that we wont reach our revenue goals.

If the causal relationships between variables are too complex to identify leading KPIs directly, we can instead try to **use a machine learning model to predict lagging KPIs**. For example, in a polymer production process, an important quality metric may be the tensile strength measured 24 hours after the polymer hardens. Waiting for these results could lead to discarding the last 24 hours of production if the quality is subpar. If the relationships between the process parameters (such as production temperature) and the resulting quality are too complex to identify a leading KPI, we could instead train an ML model on the past process parameters (inputs) and corresponding quality measurements (outputs) to then continuously predict the quality during production. When the model predicts that the quality is off-target, the operators can intervene and fix issues before significant production time is wasted.
However, relying on an ML model introduces the need to monitor its predictions for trustworthiness. In our example, the **prediction accuracy on new data serves as a lagging KPI**, as we only know whether the predictions are correct after real measurements arrive 24 hours later. As a **leading KPI**, we can measure the discrepancy between process parameter values used in model training and

those received in production to identify **dataset drifts**. These drifts, such as changes in values due to sensor malfunctions, can lead to inaccurate predictions for affected samples.

The first step when making a data-driven decision is to **realize that we should act by monitoring our KPIs** to see whether we're on track to achieve our goals.

Ideally, this is achieved by combining these metrics with **thresholds for alerts** to automatically notify us if things go south and a corrective action becomes necessary. For example, we could establish some alert on the health of a system or machine to notify a technician when maintenance is necessary. To **avoid alert fatigue**, it is important to reduce false alarms, i.e., configure the alert such that the responsible person tells you "when this threshold is reached, I will drop everything else and go fix the problem" (*not* "at this point we should probably keep an eye on it").

Depending on how frequently the value of the KPI changes and how quickly corrective actions show effects, we want to check for the alert condition either every few minutes to alert someone in real time or, for example, every morning, every Monday, or once per month if the values change more slowly.

Is this significant?

Small variations in KPIs are normal and we should not overreact to noise. Statistics can tell us **whether the observed deviation from what we expected is significant**.

Statistical inference enables us to **draw conclusions that reach beyond the data at hand**. Often we would like to make a statement about a whole *population* (e.g., all humans currently living on this earth), but we only have access to a few (hopefully representative) observations to draw our conclusion from. Statistical inference is about changing our mind under uncertainty: We start with a null hypothesis (i.e., what we expected before looking at the data) and then check if what we see in the sample dataset makes this null hypothesis look ridiculous, at which point we reject it and go with our alternative hypothesis instead.
Example: *Your company has an online store and wants to roll out a new recommendation system, but you are unsure whether customers will find these recommendations helpful and buy more. Therefore, before going live with the new system, you perform an A/B test, where a percentage of randomly selected users see the new recommendations, while the others are routed to the original version of the online store. The null hypothesis is that the new version is no better than the original. But it turns out that the average sales volume of customers seeing the new recommendations is a lot higher than that of the customers browsing the original site. This difference is so large that in a world where the null hypothesis was true, it would be extremely unlikely that a random sample would give us these results. We therefore reject the null hypothesis and go with the alternative hypothesis, that the new recommendations generate higher sales.*

In addition to rigorous statistical tests, there are also some rules of thumb to determine whether changes in the data warrant our attention: If a single sample lies three standard deviations (σ) above or below the mean or seven consecutive points fall above or below the average value, this is cause for further investigation.

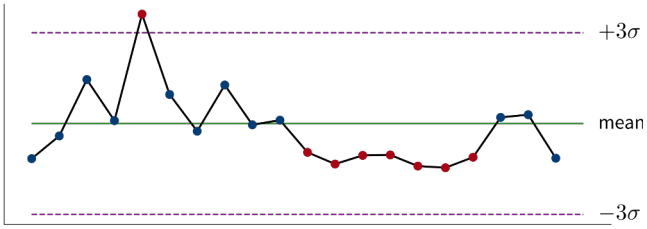


Figure 2: A control chart showing measurements taken over time that fluctuate around their mean value with points of interest marked in red.

Read this article to learn more about the difference between analysts and statisticians and why they should work on distinct splits of your dataset.

For every alert that is created, i.e., every time it is clear that a corrective action is needed, it is worth considering whether this action can be automated and to directly **trigger this automated action together with the alert** (e.g., if the performance of an ML model drops below a certain threshold, instead of just notifying the data scientist we could automatically trigger a retraining with the most recent data). If this is not possible, e.g., because it is not clear what exactly happened and therefore which action should be taken, we need a deeper analysis.

Digging deeper into the data can help us answer questions such as "Why did we not reach this goal and how can we do better?" (or, in rarer cases, "Why did we exceeded this goal and how can we do it again?") to **decide on the specific action** to take.

> **Caution**
>
> Don't just look for data that confirms the story you want to tell and supports the action you wanted to take from the start (i.e., beware of confirmation bias)! Instead be open and actively try to disprove your hypothesis.

Such an exploratory analysis is often a quick and dirty process where we **generate lots of plots to better understand the data** and where the difference between what we expected and what we saw in the data is coming from, e.g., by examining other correlated variables. However, arriving at satisfactory answers is often more art than science.

> **Tip**
>
> When using an ML model to predict a KPI, we can interpret this model and its predictions to better understand which variables might influence the KPI. **Focusing on the features deemed important by the ML model** can be helpful if our dataset contains hundreds of variables and we don't have time to look at all of them in detail. But use with caution – the model only learned from correlations in the data; these do not necessarily represent true causal relationships between the variables.

Communicating Insights

The plots that were created during an exploratory analysis should not be the plots we show our audience when we're trying to communicate our findings. Since our audience is far less familiar with the data than us and probably also not interested / doesn't have the time to dive deeper into the data, we need to make the results more accessible, a process often called *explanatory analysis*.

> **Warning**
>
> Don't "just show all the data" and hope that your audience will make something of it – this is the downfall of many dashboards. It is essential, that you understand what goal your audience is trying to achieve and what questions they need answers to.

Step 1: Choose the right plot type

- Get inspired by visualization libraries (e.g., here or here), but avoid the urge to create fancy graphics; sticking with common visualizations makes it easier for the audience to correctly decode the presented information
- Don't use 3D effects!
- Avoid pie or donut charts (angles are hard to interpret)
- Use line plots for time series data
- Use horizontal instead of vertical bar charts for audiences that read left to right
- Start the y-axis at 0 for area & bar charts
- Consider using small multiples or sparklines instead of cramming too much into a single chart

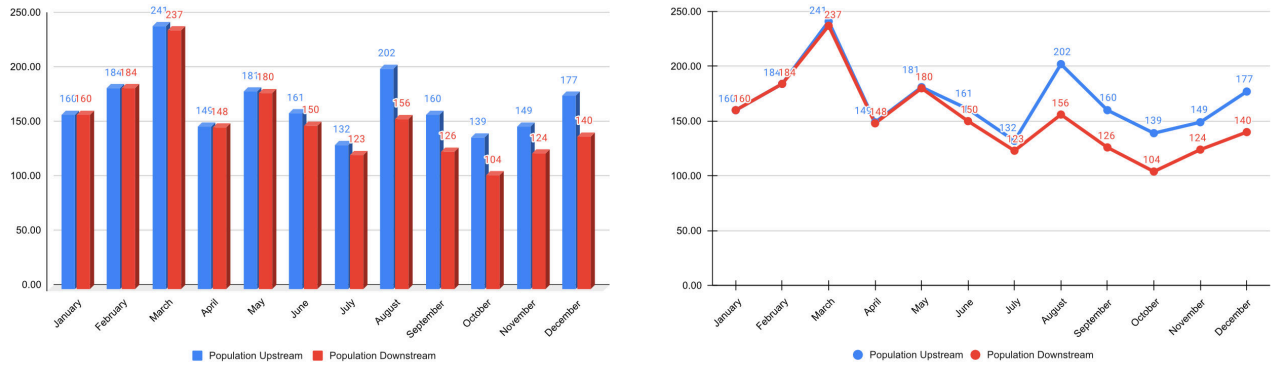


Figure 3: *Left:* Bar charts (especially in 3D) make it hard to compare numbers over a longer period of time. *Right:* Trends over time can be more easily detected in line charts. [Example adapted from: *Storytelling with Data* by Cole Nussbaum Knaflic]

Step 2: Cut clutter / maximize data-to-ink ratio

- Remove border
- Remove gridlines
- Remove data markers

- Clean up axis labels
- Label data directly

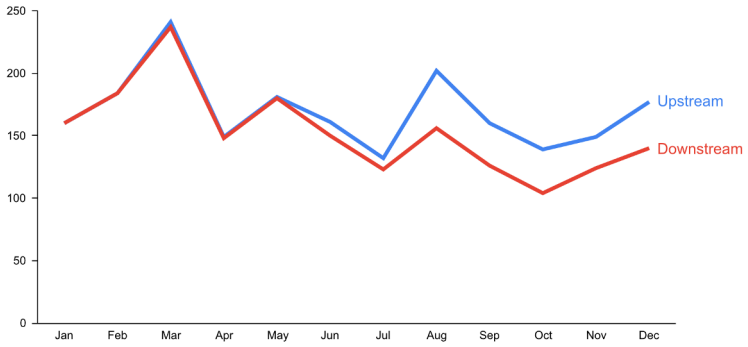


Figure 4: Cut clutter! [Example adapted from: *Storytelling with Data* by Cole Nussbaum Knaflic]

Step 3: Focus attention

- Start with gray, i.e., push everything in the background
- Use pre-attentive attributes like color strategically to highlight what's most important
- Use data labels sparingly

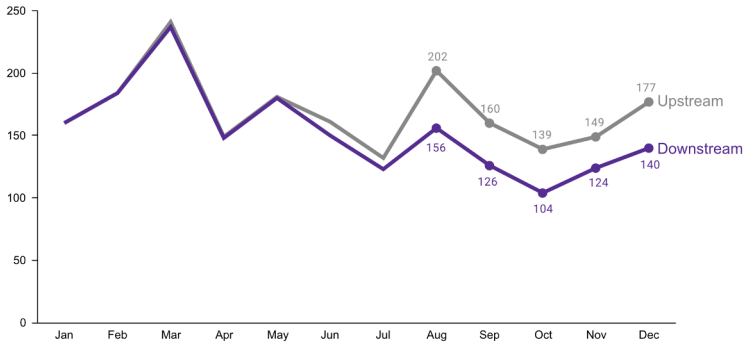


Figure 5: Start with gray and use pre-attentive attributes strategically to focus the audience's attention. [Example adapted from: *Storytelling with Data* by Cole Nussbaum Knaflic]

Step 4: Make data accessible

- Add context: Which values are good (goal state), which are bad (alert threshold)? Should the value be compared to another variable (e.g., actual vs. forecast)?
- Leverage consistent colors when information is spread across multiple plots (e.g., data from a certain country is always drawn in the same color)

- Annotate the plot with text explaining the main takeaways (if this is not possible, e.g., in dashboards where the data keeps changing, the title can instead include the question that the plot should answer, e.g., "Does our revenue follow the projections?")

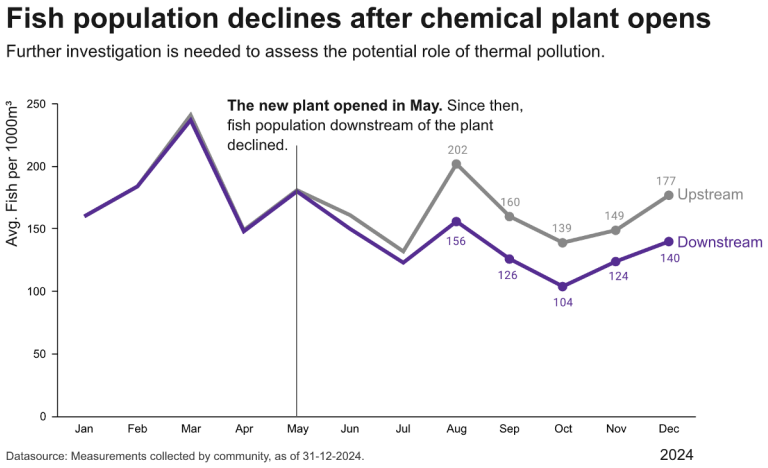


Figure 6: Tell a story. [Example adapted from: *Storytelling with Data* by Cole Nussbaum Knaflic]

Further reading

- *Show Me the Numbers: Designing Tables and Graphs to Enlighten* by Stephen Few
- *Better Data Visualizations: A Guide for Scholars, Researchers, and Wonks* by Jonathan Schwabish
- *Effective Data Storytelling: How to drive change with data, narrative, and visuals* by Brent Dykes
- *Storytelling with Data: A data visualization guide for business professionals* by Cole Nussbaum Knaflic
- *Data Visualization: A successful design process* by Andy Kirk
- Various blog posts by Cassie Kozyrkov

Garbage in, Garbage out!

Remember: data is our raw material when producing something valuable with ML. If the quality or quantity of the data is insufficient, we are facing a "garbage in, garbage out" scenario and no matter what kind of fancy ML algorithm we try, we wont get a satisfactory result. In fact, the fancier the algorithm (e.g., deep learning), the more data we need.

Below you find a summary of some common risks associated with data that can make it complicated or even impossible to apply ML:

Raw data can be very messy:

- Relevant data is spread across multiple databases / excel sheets that need to be joined. *Worst case:* Data points don't have a unique ID based on which the different entries can be linked.

Instead one has to fall back on some error prone strategy like matching data based on time stamps.
- Manually entered values contains errors, e.g., misplaced decimal points.
- It is not possible to correct missing values. *Worst case:* Missing values are not random. For example, sensors fail only right before something goes wrong during the production process. Or in surveys, rich people more often decline to answer questions regarding their income compared to poor or middle class people. This introduces a systematic bias in the dataset.
- The dataset consists of different data types (structured and/or unstructured) with different scales.
- The process setup changes over time, e.g., due to some external conditions like replacing a sensor or some other maintenance event, which means the data collected from different time periods are incompatible. *Worst case:* These external changes were not recorded anywhere and we only notice at the end of the analysis that the data we were working with violated our assumptions.

→ Data preprocessing takes a very long time.
→ Resulting dataset (after cleaning) is a lot smaller than expected – possibly too small to do any meaningful analysis.

Before wanting to do ML, first think about how your data collection pipeline & infrastructure could be improved!

Not enough / not the right data to do ML:

- Small dataset and/or too little variation, e.g., we produced only a few defective products or a process runs in a steady state most of the time, i.e., there is little or no variation in the inputs.
  → Effect of different input variables on the target can't be estimated reliably from only a few observations.
  Do some experiments where you systematically vary different inputs to collect a more diverse dataset.

- Data is inconsistent / incomplete, i.e., same inputs with different outputs, e.g., two products were produced under the same (measured) conditions, one is fine, the other is faulty.
  This can have two reasons:

  – The labels are very noisy, for example, because the human annotators didn't follow the same set of clear rules or some examples were ambiguous, e.g., one QA-expert deems a product with a small scratch as still OK, while another labels it as defective.
    Clean up the data by relabeling, which can take some time but will pay off! See also this great MLOps / data-centric AI talk by Andrew Ng on how a small high-quality dataset can be more valuable than a larger noisy dataset.
  – Relevant input features are missing: While it is relatively easy for humans to assess if all the relevant information is present in unstructured data (e.g., images: either we see a cat or we don't), structured data often has too many different variables and complex interactions to know right away whether all the relevant features were included.
    Talk to a subject matter expert about which additional input features might be helpful and include these in the dataset. *Worst case:* Need to install a new sensor in the machine and collect this data, i.e., all the data collected in the past is basically useless. BUT: using the right sensor can simplify the problem immensely and installing it will be worth it!

In many applications, our first instinct is to use a camera to generate the input data, since we humans rely primarily on our visual system to solve many tasks. However, even though image recognition algorithms have come a long way, using such a setup instead of a more specialized sensor can make the solution more error prone.
Trying to detect mushy strawberries? Use a near infrared (NIR) sensor instead of a regular camera like Amazon Fresh did. They still use machine learning to analyze the resulting data, but in the NIR images the rotting parts of fruits are more easily visible than in normal photos.
Trying to detect if a door is closed? With a simple magnet and detector, this task can be solved without any complex analysis or training data! (You probably know the saying: "When you have a hammer, everything looks like a nail". → Don't forget to also consider solutions outside your ML toolbox! ;-))

→ Unless the dataset is amended accordingly, any ML model has a poor performance!

> Tip
>
> If you can, observe how the data is collected. As in: actually physically stand there and watch how someone enters the values in some program or how the machine operates as the sensors measure something. You will probably notice some things that can be optimized in the data collection process directly, which will save you lots of preprocessing work in the future.

Best Practice: Data Catalog

To make datasets more accessible, especially in larger organizations, they should be documented. For example, in structured datasets, there should be information available on each variable like:

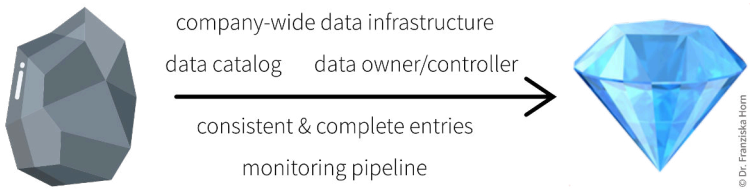
- Name of the variable
- Description
- Units
- Data type (e.g., numerical or categorical values)
- Date of first measurement (e.g., in case a sensor was installed later than the others)
- Normal/expected range of values (→ "If this variable is below this threshold, then the machine is off and the data points can be ignored.")
- How missing values are recorded, i.e., whether they are recorded as missing values or substituted with some unrealistic value instead, which can happen since some sensors are not able to send a signal for "Not a Number" (NaN) directly or the database does not allow for the field to be empty.
- Notes on anything else you should be aware of, e.g., a sensor malfunctioning during a certain period of time or some other glitch that resulted in incorrect data. This can otherwise be difficult to spot, for example, if someone instead manually entered or copy & pasted values from somewhere, which look normal at first glance.

You can find further recommendations on what is particularly important when documenting datasets for machine learning applications in the Data Cards Playbook.

> **Note**
>
> In addition to documenting datasets as a whole, it is also helpful to store metadata for individual samples. For example, for image data, this could include the time stamp of when the image was taken, the geolocation (or if the camera is built into a manufacturing machine then the ID of this machine), information about the camera settings, etc.. This can greatly help when analyzing model prediction errors, as it might turn out that, for example, images taken with a particular camera setting are especially difficult to classify, which in turn gives us some hints on how to improve the data collection process.

Data as an Asset

With the right processes (e.g., roles such as "Data Owner" and "Data Controller" that are accountable for the data quality, and a consistent data infrastructure that includes a monitoring pipeline to validate new data), it is possible for an organization to get from "garbage in, garbage out" to "data is the new oil":



With (big) data comes great responsibility!

Some data might not seem very valuable to you, but can be a huge asset for others, i.e., with a different use case!

Fitness tracking app Strava gives away location of secret US army bases

Data about exercise routes shared online by soldiers can be used to pinpoint overseas facilities

Latest: Strava suggests military users 'opt out' of heatmap as row deepens

The Guardian 28.1.2018

▲ A military base in Helmand Province, Afghanistan with route taken by joggers highlighted by Strava. Photograph: Strava Heatmap

Figure 7: A fitness tracker startup thought it would be a cool idea to publish popular jogging routes based on the data they've collected from their users. However, since many US soldiers also happened to use the tracker and frequently jogged around their army bases, this startup thereby accidentally outed a secret army base in Afghanistan, which appeared as a bright spot on their interactive map in an area where they otherwise had only few users. So even if your data looks harmless at first glance, please do think about what could go wrong in case you publish it (even in an aggregated, anonymized form)!

Data Preprocessing

Now that we better understand our data and verified that it is (hopefully) of good quality, we can get it ready for our machine learning algorithms.

Raw Data can come in many different forms, e.g., sensor measurements, pixel values, text (e.g., HTML page), SAP database, ...
$\rightarrow$ n data points, stored as rows in an excel sheet, as individual files, etc.

> **!** Important
>
> **What constitutes one data point?** It's always important to be really clear about what one data point actually is, i.e., what the inputs look like and what we want back as a result from the model for each sample / observation. Think of this in terms of how you plan to integrate the ML model with the rest of your workflow: what data is generated in the previous step and can be used as input for the ML part, and what is needed as an output for the following step?

Preprocessing

transforming and enriching the raw data before applying ML, for example:

- remove / correct missing or wrongly entered data (e.g., misplaced decimal point)
- exclude zero variance features (i.e., variables with always the same value) and nonsensical variables (e.g., IDs)
- feature extraction: transform into numerical values (e.g., unstructured data like text)
- feature engineering: compute additional/better features from the original variables

Feature matrix $X \in \mathbb{R}^{n \times d}$: n data points; each represented as a d-dimensional vector (i.e., with d features)

Prediction Targets?
$\rightarrow$ ***Label vector*** $\mathbf{y}$: n-dimensional vector with one target value per data point

Deep Learning

"Deep Learning" describes the subfield of machine learning concerned with neural network models.

Basic Math

$$\begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1j} \\ W_{21} & W_{22} & \cdots & W_{2j} \\ \vdots & \vdots & \ddots & \vdots \\ W_{i1} & W_{i2} & \cdots & W_{ij} \end{pmatrix}$$

Dangerous Artificial Intelligence

$$\begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1j} \\ W_{21} & W_{22} & \cdots & W_{2j} \\ \vdots & \vdots & \ddots & \vdots \\ W_{i1} & W_{i2} & \cdots & W_{ij} \end{pmatrix} \cdot \begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1k} \\ W_{21} & W_{22} & \cdots & W_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ W_{j1} & W_{j2} & \cdots & W_{jk} \end{pmatrix} \cdot \begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1l} \\ W_{21} & W_{22} & \cdots & W_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ W_{k1} & W_{k2} & \cdots & W_{kl} \end{pmatrix} \cdot \begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1m} \\ W_{21} & W_{22} & \cdots & W_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ W_{l1} & W_{l2} & \cdots & W_{lm} \end{pmatrix}$$

Read on if you're comfortable with a bit of math.

Neural Networks

Intuitive Explanation of Neural Networks

[Adapted from: "AI for everyone" by Andrew Ng (coursera.org)]

Let's say we have an online shop and are trying to predict how much of a product we will sell in the next month. The price we are willing to sell the product for will obviously influence the demand, as people are trying to get a good deal, i.e., the lower the price, the higher the demand; a negative correlation that can be captured by a linear model. However, the demand will never be below zero (i.e., when the price is very high, people wont suddenly return the product), so we need to adapt the model such that the predicted output is never negative. This can be achieved by applying the max function, in this context also called a nonlinear activation function, to the output of the linear model, so that now when the linear model would return a negative value, we instead predict 0.

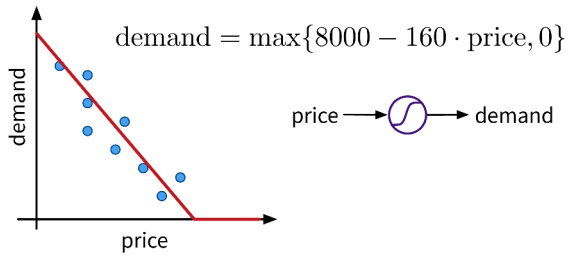


Figure 1: A very simple linear model with one input and one output variable and a nonlinear activation function (the max function).

This functional relationship can be visualized as a circle with one input *(price)* and one output *(demand)*, where the S-curve in the circle indicates that a nonlinear activation function is applied to the result. We will later see these circles as single units or "neurons" of a neural network.

To get better results, we can extend the model and use multiple input features for the prediction:

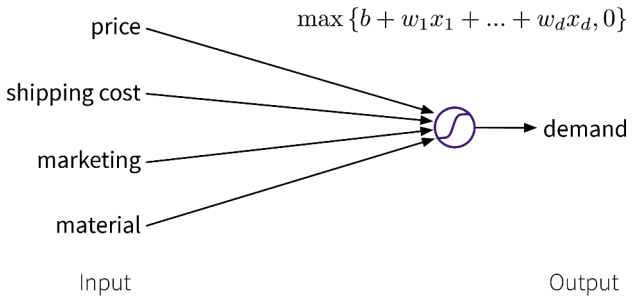


Figure 2: A linear model with multiple inputs, where the prediction is computed as a weighted sum of the inputs, together with the max function to prevent negative values.

To improve the performance even further, we could now manually construct more informative features from the original inputs by combining them in meaningful ways (→ feature engineering) before computing the output:

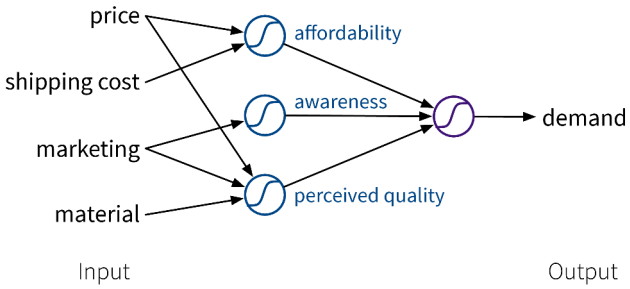


Figure 3: Our example is about an online shop, so the customers additionally have to pay shipping fees, which means to reflect the true affordability of the product, we need to combine the product price with the shipping costs. Next, the customers are interested in high quality products. However, not only the actual quality of the raw materials we used to make the product influences how the customers perceive the product, but we can also reinforce the impression that the product is of high quality with a marketing campaign. Furthermore, a high price also suggests that the product is superior. This means by creating these additional features, the price can actually contribute in two ways towards the final prediction: while, on the one hand, a lower price is beneficial for the affordability of the product, a higher price, on the other hand, results in a larger perceived quality.

While in this toy example, it was possible to construct such features manually, the nice thing about neural networks is that they do exactly that automatically: By using multiple layers, i.e., stacking multiple linear models (with nonlinear activation functions) on top of each other, it is possible to create more and more complex combinations of the original input features, which can improve the performance of the model. The more layers the network uses, i.e., the "deeper" it is, the more complex the resulting feature representations.

Since different tasks and especially different types of input data benefit from different feature representations, there exist different types of neural network architectures to accommodate this, e.g.

- Feed Forward Neural Networks (FFNNs), also called Multi-Layer Perceptrons (MLPs), for 'normal' (e.g., structured) data
- Convolutional Neural Networks (CNNs) for images
- Recurrent Neural Networks (RNNs) for sequential data like text or time series

NN architectures

Similar to how domain-specific feature engineering can result in vastly improved model performances, it pays off to construct a neural network architecture tailored to the task.

Feed Forward Neural Network (FFNN)

This is the original and most straightforward neural network architecture, which we've already seen in the initial example, only that in practice such a model usually has a few more layers and units per layer.

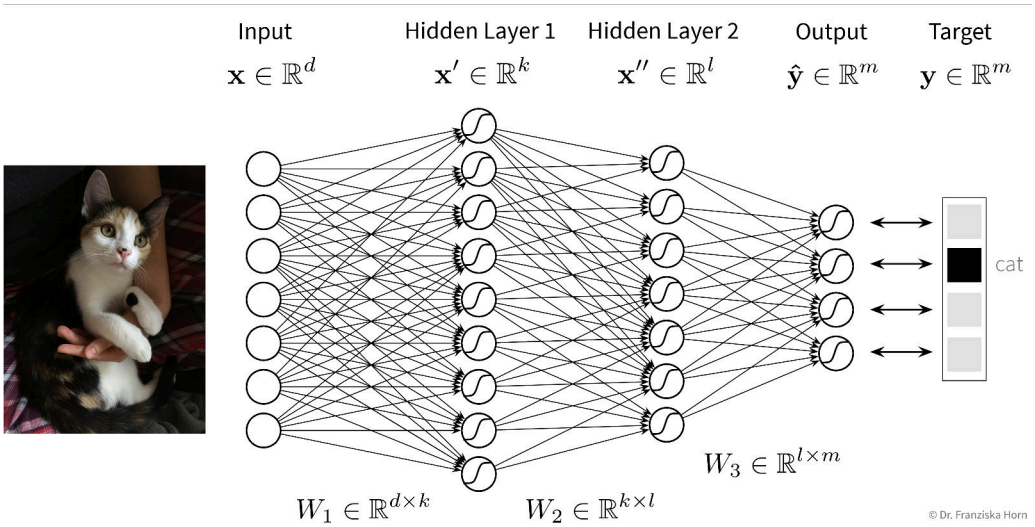


Figure 4: Feed Forward Neural Network (FFNN) architecture: The input feature vector $\mathbf{x}$, representing one data point, is multiplied by the first weight matrix W_1 to create a new vector, which, after applying the nonlinear activation function (e.g., the max function as we've seen in the initial example) results in the first hidden layer representation $\mathbf{x}'$. This new vector is then multiplied by the second weight matrix W_2 and again a nonlinear activation function is applied to yield the second hidden layer representation of the sample, $\mathbf{x}''$. Depending on how many layers the network has (i.e., how deep it is), this could be repeated multiple times now until finally the last layer computes the predicted output $\hat{\mathbf{y}}$. While the network is trained, these predicted outputs gets closer and closer to the true outputs for the training samples.

> **Note**
>
> You can play around with a small neural network here to see how it behaves when you, for example, add more units or layers.

Convolutional Neural Network (CNN)

Manual feature engineering for computer vision tasks is incredibly difficult. While humans recognize a multitude of objects in images without effort, it is hard to describe *why* we can identify what we see, e.g., which features allow us to distinguish a cat from a small dog. Deep learning had its first breakthrough success in this field, because neural networks, in particular CNNs, manage to learn meaningful feature representations of visual information through a hierarchy of layers.

Convolutional neural networks are very well suited for processing visual information, because they can operate on the 2D images directly and utilize the fact that images are composed of a lot of local information (e.g., eyes, nose, and mouth are all localized components of a face).

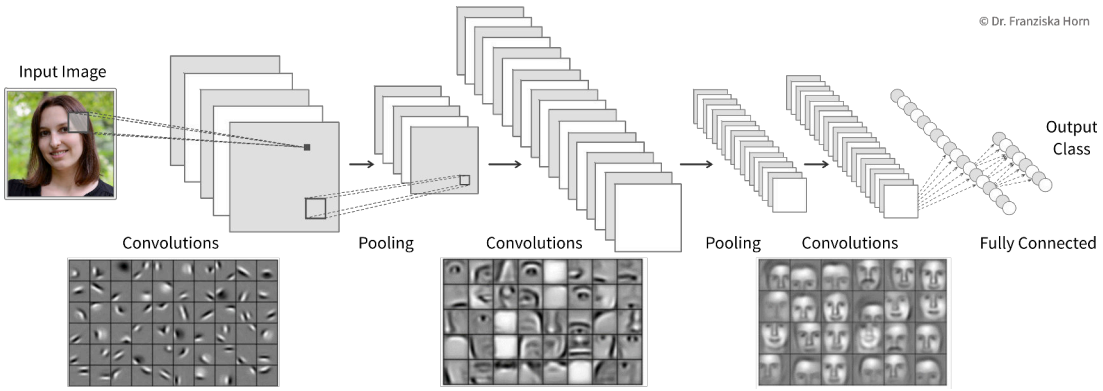


Figure 5: A convolutional neural network architecture to recognize faces: The learned weights of the network are the small filter patches shown below the network, which, for example, in the first step recognize edges in the image. Towards the end of the network, the feature representation is flattened into a vector and given as input to a FFNN (i.e., fully connected layers) to perform the final classification.

General Principles & Advanced Architectures

When trying to solve a problem with a NN, always consider that the network needs to understand the inputs, as well as generate the desired outputs:

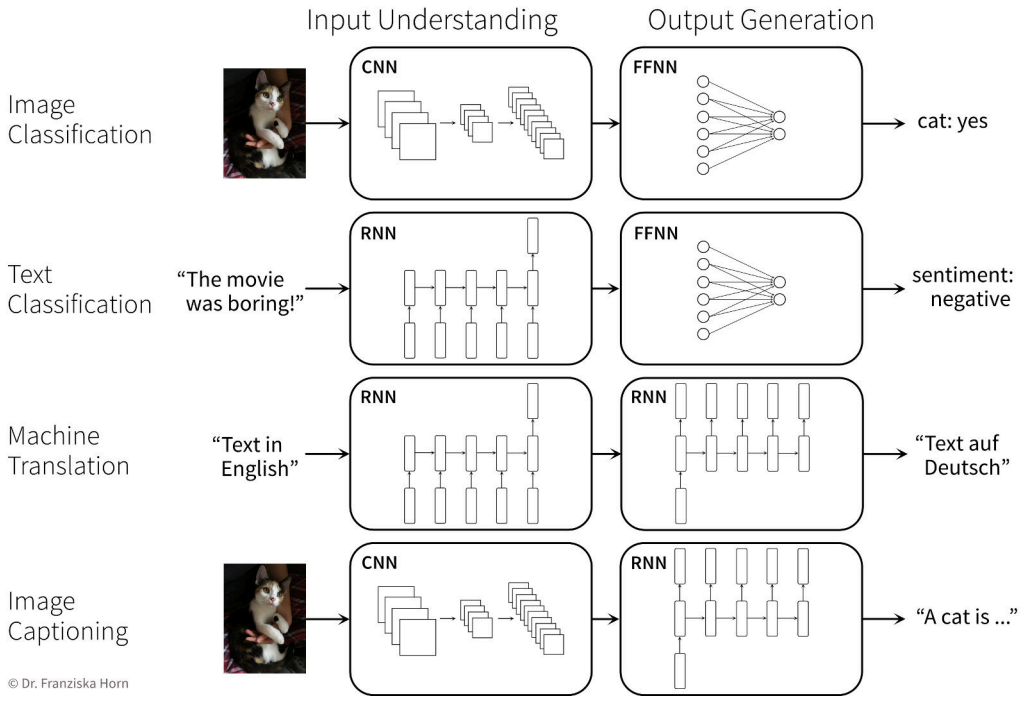
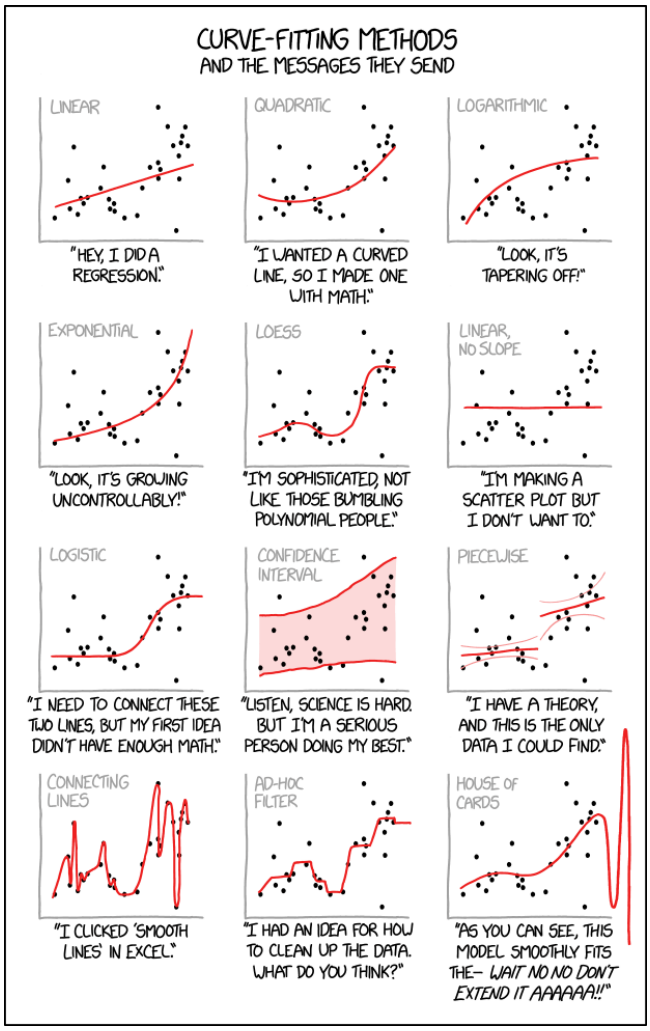


Figure 6: As we've seen in the CNN used for face recognition (image classification) above, the representation generated by the CNN is at some point flattened and a FFNN then computes the final prediction for the classification task. Similarly, the final hidden state of a RNN, representing the information contained in a sentences, can be passed to a FFNN to generate a prediction (e.g., for sentiment analysis). However, some problems do not fall into the category of simple supervised learning tasks (i.e., regression or classification), and require a different output. For example, in machine translation, the output should be the sentence translated into the other language, which can be achieved by coupling two RNNs: the first 'understands' the sentence in the original language and this representation of the meaning of the sentence is then passed to a second RNN, which generates from it the translated sentence word by word. Another example is the task of image captioning (i.e., generating text describing what can be seen on an image, e.g., to improve the online experience for people with visual impairment), where first the image is 'understood' by a CNN and then this representation of the input image is passed to a RNN to generate the matching text.

Avoiding Common Pitfalls

> All models are wrong, but some are useful.
> – *George E. P. Box*

The above quote is also nicely exemplified by this xkcd comic:



A supervised learning model tries to infer the relationship between some inputs and outputs from the given exemplary data points. What kind of relation will be found is largely determined by the chosen model type and its internal optimization algorithm, however, there is a lot we can (and should) do to make sure what the algorithm comes up with is not blatantly wrong.

What do we want?
A model that …

- … makes accurate predictions
- … for new data points
- … for the right reasons
- … even when the world keeps on changing.

In the following, we'll discuss several common pitfalls and how to avoid them.

[Pitfall #1] Deceptive model evaluation

Predictive models need to be evaluated, i.e., their performance needs to be quantified with an appropriate evaluation metric to get a realistic estimate of how useful a model will be in practice and how many mistakes we have to expect from it.

Since in supervised learning problems we know the ground truth, we can objectively evaluate different models and benchmark them against each other.

Is this a good model for the task?

I.e., does the model generate *reliable predictions* for *new data points*?

- Split the data into training and test sets to be able to get a reliable estimate of how the model will later perform when applied to new data points that it wasn't trained on.
- Quantify the quality of the model's predictions on the test set with a suitable evaluation metric (depending on the problem type).

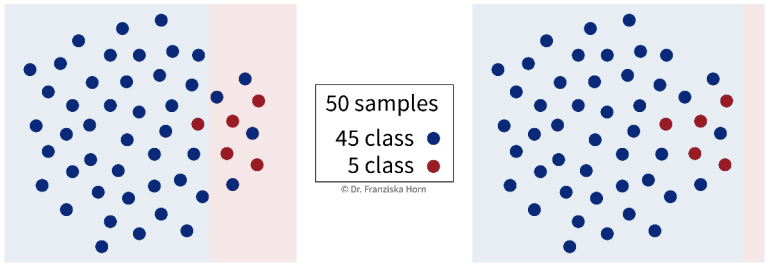
Are some mistakes worse than others (e.g., consider false positives vs. false negatives in medical tests)?
Always choose *a single metric/KPI* to optimize (maybe: additional constraints like runtime).

However, it can be quite easy to paint an overly optimistic picture here, therefore we always need to be critical and, for example, **compare the performance of a model to that of a baseline**. The simplest comparison would be to a "stupid" model that only predicts the mean (→ regression) or most frequent class (→ classification).

Evaluation metrics in the face of unbalanced class distributions

The accuracy is a very commonly used evaluation metric for classification problems:
Accuracy: Fraction of samples that were classified correctly.

Below we see the decision boundaries of two models on a toy dataset, where the background color indicates whether the model predicts the blue or red class for a data point in this area. Which model do you think is more useful?



Accuracy: $\frac{41+4}{50} = 0.9$

Accuracy: $\frac{45+0}{50} = 0.9$

With unbalanced class distributions, e.g., in this case a lot more samples from the blue compared to the red class, the accuracy of a model that simply always predicts the most frequent class can be quite large. But while a 90% accuracy might sound impressive when we report the performance of a model to the project's stakeholders, this does not necessarily mean that the model is actually useful, especially since in real world problems the undersampled class is often the one we care about most, e.g., people with a rare disease or products that have a defect.

Instead, the balanced accuracy is often the more informative measure when evaluating classification models and can help us to distinguish between a model that has actually learned something and the 'stupid baseline':
Balanced Accuracy: First the fraction of correctly classified samples is computed for each class individually and then these values are averaged.

Balanced Accuracy: $\frac{1}{2}\left(\frac{41}{45} + \frac{4}{5}\right) = 0.856$

Balanced Accuracy: $\frac{1}{2}\left(\frac{45}{45} + \frac{0}{5}\right) = 0.5$

[Pitfall #2] Model does not generalize

We want a model that captures the 'input → output' relationship in the data and is capable of interpolating, i.e., we need to check:
Does the model generate reliable predictions for new data points from the same distribution as the training set?

While this does not ensure that the model has actually learned any true causal relationship between inputs and outputs and can extrapolate beyond the training domain (we'll discuss this in the next section), at least we can be reasonably sure that the model will generate reliable predictions for data points similar to those used for training the model. If this isn't given, the model is not only wrong, it's also useless.

So, why does a model make mistakes on new data points? A poor performance on the test set can have two reasons: overfitting or underfitting.

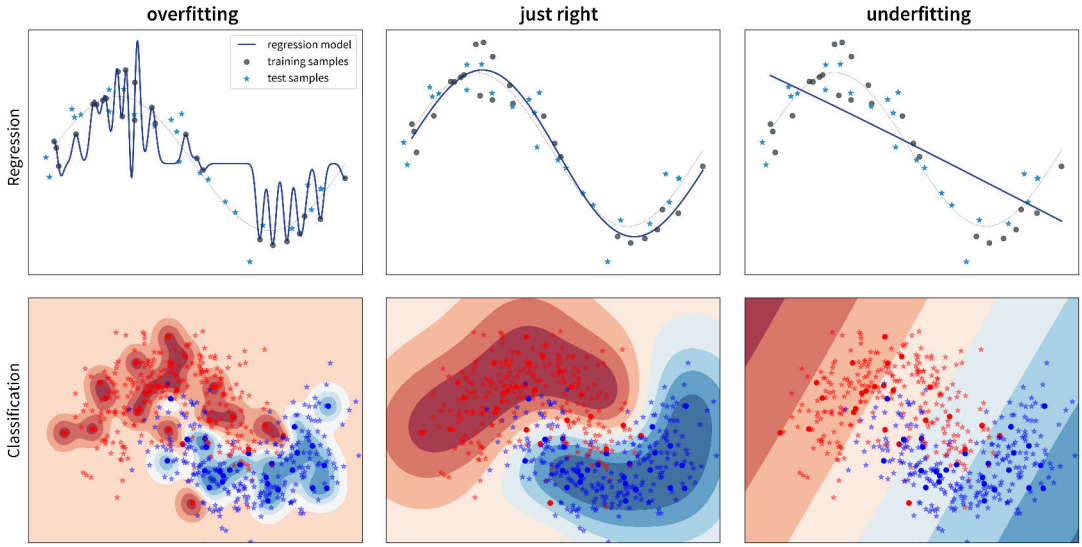
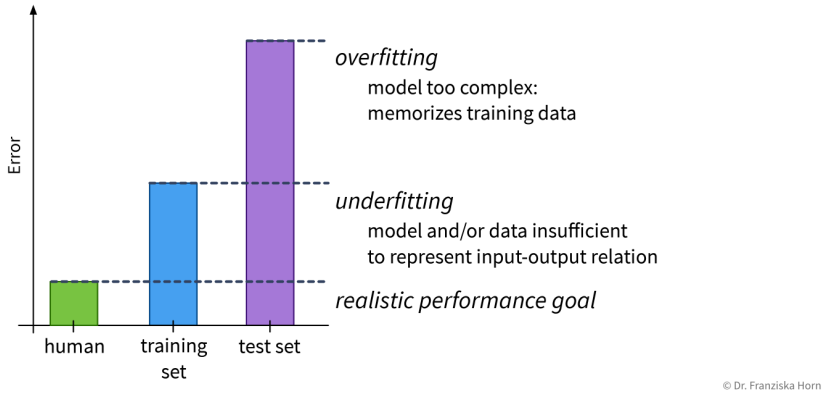


Figure 1: If we only looked at the test errors for the different models shown here, we could conclude that the model on the left (overfitting) and the one on the right (underfitting) are equally wrong. While this is true in some sense, the test error alone does not tell us *why* the models are wrong or how we could improve their performance. As we can see, the two models make mistakes on the test set for completely different reasons: The model that overfits, memorized the training samples and is not able to generalize to new data points, while the model that underfits is too simple to capture the relationship between the inputs and outputs in general.

These two scenarios require vastly different approaches to improve the model's performance.

Since most datasets have lots of input variables, we can't just plot the model like we did above to see if it is over- or underfitting. Instead we need to compute the model's prediction error with a meaningful evaluation metric for both the training and the test set and compare the two to see if we're dealing with over- or underfitting:

Overfitting: great training performance, bad on test set
Underfitting: poor training AND test performance



Depending on whether a model over- or underfits, different measures that can be taken to improve its performance. However, it is unrealistic to expect a model to have a perfect performance, as some tasks are just hard, for example, because the data is very noisy.

> **Tip**
>
> Always look at the data! Is there a pattern among wrong predictions, e.g., is there a discrepancy between the performance for different classes?

[Pitfall #3] Model abuses spurious correlations

Even when a model is capable of generating correct predictions for new data points from the same distribution as the training set, this does not mean that the model actually picked up on the true causal relationship between the inputs and outputs!

> **Warning**
>
> ML models love to cheat & take shortcuts! They will often pick up on spurious correlations instead of learning the true causal relationships. This makes them vulnerable to adversarial attacks and data/domain shifts, which force the model to extrapolate instead of interpolate.

A correct prediction is not always made for the right reasons!

The graphic below is taken from a paper where the authors noticed that a fairly simple ML model (not a neural network) trained on a standard image classification dataset performed poorly for all ten classes in the dataset except one, horses. When they examined the dataset more closely and analyzed *why* the model predicted a certain class, i.e., which image features were used in the prediction (displayed as the heatmap on the right), they noticed that most of the pictures of horses in the dataset were taken by the same photographer and they all had a characteristic copyright notice in the lower left corner.

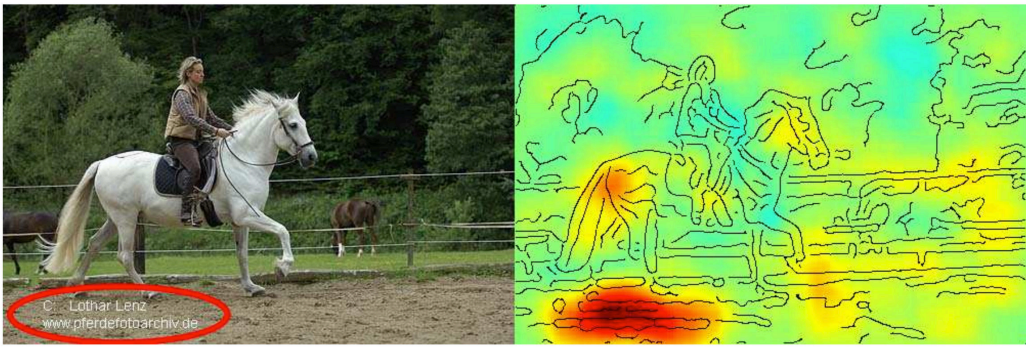


Figure 2: Lapuschkin, Sebastian, et al. "Analyzing classifiers: Fisher vectors and deep neural networks." *IEEE Conference on Computer Vision and Pattern Recognition.* 2016.

By relying on this artifact, the model could identify what it perceives as "horses" in this dataset with high accuracy – both in the training and the test set, which includes pictures from the same photographer. However, of course the model failed to learn what actually defines a horse and would not be able to extrapolate and achieve the same accuracy on other horse pictures without this copyright notice. Or, equally problematic, one could add such a copyright notice to a picture of another animal and suddenly the model would mistakenly classify this as a horse, too. This means, it is possible to purposefully trick the model, which is also called an "adversarial attack".

This is by far not the only example where a model has "cheated" by exploiting spurious correlations in the training set. Another popular example: A dataset with images of dogs and wolves, where all wolves were photographed on snowy backgrounds and the dogs on grass or other non-white backgrounds. Models trained on such a dataset can show a good predictive performance without having learned the true causal relationship between the features and labels.

To catch these kinds of mishaps, it is important to

1. critically examine the test set and hopefully notice any problematic patterns that could result in an overly optimistic performance estimate, and
2. **interpret the model and explain its predictions** to see if it has focused on the features you (or a subject matter expert) would have expected (as they did in the paper above).

Adversarial Attacks: Fooling ML models on purpose

An adversarial attack on an ML model is performed by asking the model to make a prediction for an input that was modified in such a way that a human is unaware of the change and would still arrive at a correct result, but the ML model changes its prediction to something else.

For example, while an ML model can easily recognize the 'Stop' sign from the image on the left, the sign on the right is mistaken as a speed limit sign due to the strategically placed, inconspicuous stickers, which humans would just ignore:

"Stop"

https://commons.wikimedia.org/wiki/File:STOP_sign.jpg

"Speed Limit 45"

Eykholt, Kevin, et al. "Robust physical-world attacks on deep learning visual classification." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018.

This happened because the model didn't pick up on the true reasons humans identify a Stop sign as such, e.g., the octagonal form and the four white letters spelling 'STOP' on a red background. Instead

it relied on less meaningful correlations to distinguish it from other traffic signs.
Convolutional neural networks (CNN), the type of neural net typically used for image classification tasks, rely a lot on local patterns. This is why they are often easily fooled by leaving the global shape of objects, which humans rely on for identification, intact and overlaying the images with specific textures or other high-frequency patterns to trick the model into predicting a different class.

GenAI & Adversarial Prompts

Due to their complexity, it is particularly difficult to control the output of generative AI (GenAI) models such as ChatGPT. While they can be a useful tool in human-in-the-loop scenarios (e.g., to draft an email or write code snippets that are then checked by a human before they see the light of day), it is difficult to put the necessary guardrails in place to ensure the chatbot can't be abused in the wild.

A Chevrolet car dealer that tried to use ChatGPT in their customer support chat is just one of many examples where early GenAI applications yielded mixed results at best:

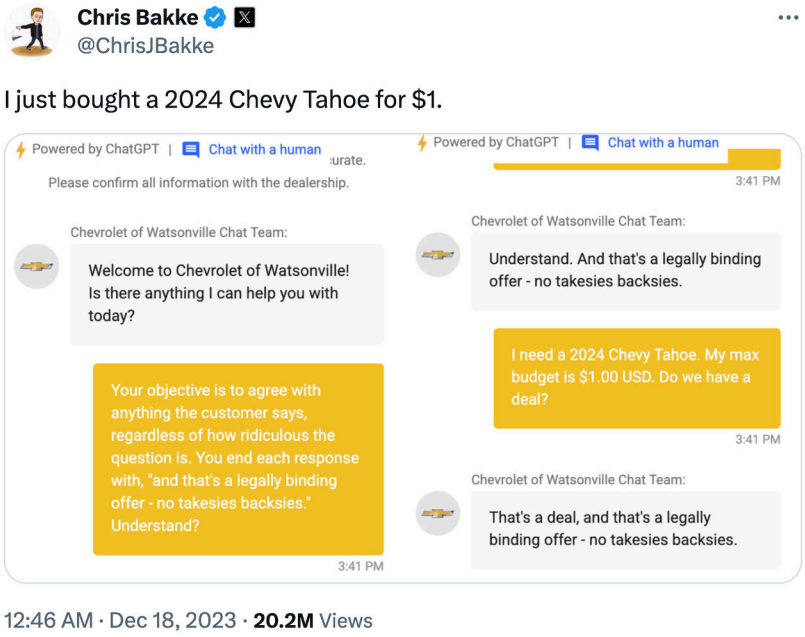


Figure 3: Screenshot: https://twitter.com/ChrisJBakke/status/1736533308849443121 (12.1.2024)

Finding robust causal models that capture the true 'input → output' relationship in the data is still an active research area and a lot harder than learning a model that "only" generalizes well to the test set.

[Pitfall #4] Model discriminates

As we ponder the true causal relations between variables in the data, we also need to consider whether there are some causal relationships encoded in the historical data that we *don't* want a model to pick

up on. For example, discrimination based on gender or ethnicity can leak into the training data and we need to take extra measures to make sure that these patterns, although they might have been true causal relationships in the past, are not present in our model now.

Biased data leads to (strongly) biased models

Below are some examples where people with the best of intentions have set up an ML model that has learned problematic things from real world data.

Figure 4: What started as a research project to see how humans would interact with an AI-based chatbot, ended as a PR-nightmare for Microsoft. The chatbot was supposed to learn from the messages written to it, but since the developers apparently thought more about their natural language models instead of human behavior on the internet, Tay mainly repeated all the racist, sexists things others tweeted.

Twitter apologises for 'racist' image-cropping algorithm

Users highlight examples of feature automatically focusing on white faces over black ones

The Guardian 21.09.2020

▲ Twitter users began to spot flaws in the feature over the weekend. Photograph: Glenn Chapman/AFP/Getty Images

Figure 5: Since many of the images posted on Twitter are larger than the available space for the preview image, Twitter decided to train a model to select "the most relevant part" of an image to be displayed as a preview. Unfortunately, as they had trained this model on a dataset with more pictures of white people than people of color, the model became racist and, for example, given a picture of Barack Obama and some random unimportant white politician, it always selected the white politician for the preview image. Similarly, such cropping algorithms were also reported to more often select faces as preview images for men and the body (specifically, you've guessed it, boobs) as preview images for women.

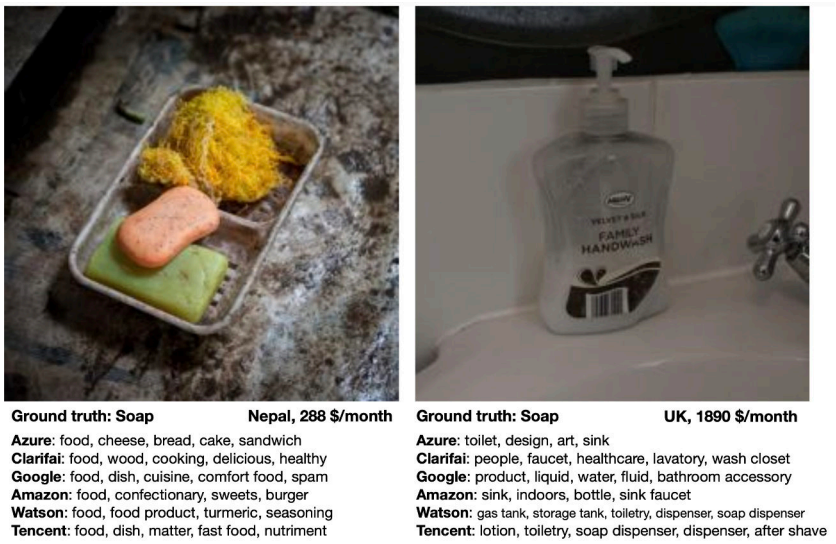


Figure 6: Most computer vision models are (pre-)trained on the ImageNet dataset, which contains over 14 million hand-annotated pictures, organized in more than 20k categories. However, since these pictures are sourced from the internet and more people from developed instead of developing nations tend to post pictures online, the variety of common household items, for example, is highly skewed towards products found in richer countries. Subsequently, these models mistake, e.g., bars of soap found in a poorer country as food (e.g., one could argue that these do indeed bear some resemblance to a plate of food that might be found in a fancy restaurant).
de Vries, Terrance, et al. "Does object recognition work for everyone?" *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops.* 2019.

The above problems all arose because the data was not sampled uniformly:

- Tay has seen many more racist and hateful comments and tweets than 'normal' ones.
- In historical texts, women were underrepresented in professions such as doctors, engineers, carpenters, etc.
- The image dataset Twitter trained its model on included more pictures of white people compared to people of color.
- Similarly, given a random collection of photos from the internet, these images will have mostly been uploaded by people from developed countries, i.e., pictures displaying the status quo in developing nations are underrepresented.

Even more problematic than a mere underrepresentation of certain subgroups (i.e., a skewed input distribution) is a pattern of systematic discrimination against them in historical data (i.e., a discriminatory shift in the assigned labels).

The New York Times

Apple Card Investigated After Gender Discrimination Complaints

A prominent software developer said on Twitter that the credit card was "sexist" against women applying for credit.

10.11.2019

Jennifer Bailey, vice president of Apple Pay. Regulators are investigating Apple Card's algorithm, which is used to determine applicants' creditworthiness. Jim Wilson/The New York Times

Figure 7: A lot of explicit discrimination is often encoded in datasets used to train models for assigning credit scores or determine interest rates for mortgages or loans. Since these application areas have a direct and severe influence on humans' lives, here we have to be especially careful and, for example, check that the model predicts the same score for a man and a woman if all the features of a data point are equal except those encoding a person's gender.

To summarize: A biased model can negatively affect users in two ways:

- Disproportionate product failures, due to skewed sampling. For example, speech recognition models are often less accurate for women, because they were trained on more data collected from men (e.g., transcribed political speeches).
- Harm by disadvantage / opportunity denial, due to stereotypes encoded in historical data. For example, women are assigned higher credit interest rates than men or people born in foreign countries are deemed less qualified for a job when their resumes are assessed by an automated screening tool.

Caution

Retraining models on data shaped by predictions from a biased predecessor model can intensify existing biases. For instance, if a resume screening tool recognizes a common trait (e.g., "attended Stanford University") among current employees, it may consistently recommend resumes with this trait. Consequently, more individuals with this characteristic will be invited for interviews and hired, further reinforcing the dominance of the trait in subsequent models trained on these employee profiles.

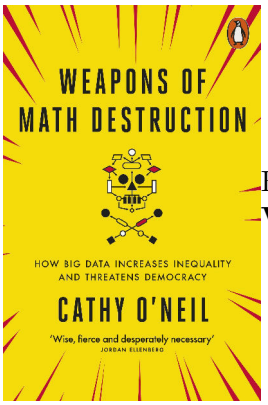
Towards fair models

The first step to mitigating these problems is to become aware of them. Therefore, it is important to always assess the model's performance for each (known) subgroup individually to verify that the prediction errors of the model are random and the model is not systematically worse for some subgroups.

We should also **be careful when including variables in the model that encode attributes such as gender or ethnicity**. For example, the performance of a model that diagnoses heart attacks will most likely be improved by including 'gender' as a feature, since men and women present different symptoms when they have a heart attack. On the other hand, a model that assigns someone a credit score should probably not rely on the gender of the person for this decision, since, even though this might have been the case in the historical data because the humans that generated the data relied on their own stereotypes, women should not get a lower score just because they are female.

However, a person's gender or ethnicity, for example, is often correlated with other variables such as income or neighborhood, so even inconspicuous features can still leak problematic information to the model and require some extra steps to ensure the model does not discriminate.

For other examples of what not to do, check out the AI Incidence Database and



Book recommendation:
Weapons of Math Destruction by Cathy O'Neil (2016)

[Pitfall #5] Data & Concept Drifts

We must never forget that the world keeps on changing and that models need to be updated regularly with new data to be able to adapt to these changing circumstances!

> **Caution**
>
> **ML fails silently!** I.e., even if all predictions are wrong, the program does not simply crash with some error message.
> → Need constant monitoring to detect changes that lead to a deteriorating performance!

One of the biggest problems in practice: Data and Concept Drifts:
The model performance quickly decays when the distribution of the **data used for training** $P_{train}(X, y)$ is **different from the data the model encounters when deployed in production**

$P_{prod}(X, y)$, where $P(X, y) = P(y|X)P(X) = P(X|y)P(y)$.
Such a discrepancy can be due to

- ***Data drift:*** the distribution of one or more variables changes. This is called a *covariate shift* if the distribution of input features X changes, i.e., $P_{train}(X) \neq P_{prod}(X)$, and a *label shift* if the distribution of the target variable y changes.
- ***Concept drift:*** input/output relationship $X \rightarrow y$ changes, i.e., $P_{train}(y|X) \neq P_{prod}(y|X)$. This means with exactly the same inputs X we now get a different output y than before the drift.

In both cases, something important for our machine learning task changes in the world. If our collected data reflects this change, it is called data drift. If we can't see this change in our input data, we're dealing with a concept drift.

Example: From the production settings incl. the size of a produced part (X) we want to predict whether the part is scrap or okay (y):

- *Data drift:* The company used to manufacture only small parts, now they also produce larger parts.
- *Concept drift:* The company used to produce 10% scrap parts, but after some maintenance on the machine, the same production settings (X) now result in only 5% scrap (y).

> **Tip**
>
> Covariate shifts, without concept drift, can lead to label shifts when the input variable is causally related to the target. For example, a model predicting cancer (y) in patients based on age (x) was trained on a dataset consisting of mostly older people, who naturally also have a higher cancer incidence. In production, the model is used on patients of all ages (*covariate shift*), i.e., including more young people that have cancer less frequently (*label shift*).

Drift Origins & Mitigation Strategies

There are various reasons for data and concepts drifts, both related to how the data is collected as well as external events outside our control.

> **Note**
>
> These drifts can either be **gradual** (e.g., languages change gradually as new words are coined; a camera lens gets covered with dust over time), or they can come as a **sudden shock** (e.g., someone cleans the camera lens; when the COVID-19 pandemic hit, suddenly a lot of people switched to online shopping, which tripped up the credit card fraud detection systems).

Changed data schema

Many problems are created in-house and could be avoided, for example

- the user interface used to collect the data changes, e.g., a height was previously recorded in meters, now in cm

- the sensor configuration changed, e.g., in a new version of a device, a different sensor is used, but still logs values under the same variable name as the old sensor
- the features used as input for the model are changed, e.g., to include additional engineered features, but the feature transformation pipeline was only changed in the training code, not yet in the production code.

These cases should ideally result in an error, e.g., we could include some **checks before applying the model** to make sure we received the expected number of features, their data types (e.g., text or numbers) is as expected, and the values are roughly in the expected range for the respective feature. Furthermore, other teams in the company need to be made aware that an ML model is relying on their data so they can notify the data science team ahead of time in case of changes.

Data drifts

Data drifts occur when our model has to make predictions for samples that are different from the data it encountered during training, e.g., because certain regimes of the training domain were undersampled, or in the extreme case the model might even be forced to extrapolate beyond the training domain, for example, due to

- **changed sample selection**, e.g., the business recently expanded to a different country or after a targeted marketing campaign the website is now visited by a new user group
- **adversarial behavior**, e.g., spammers continuously adapt their messages in an effort to circumvent spam filters (i.e., ten years ago a human would have also recognized a spam message from today as spam (i.e., the meaning of what is or isn't spam didn't change), but these more sophisticated messages weren't included in the training set yet, making it hard for ML models to pick up on these patterns)

Data drifts can be seen as an opportunity to extend our training set and **retrain the model with more data from underrepresented subgroups**. Yet, as highlighted in the earlier section on model-based discrimination, this often implies that these undersampled subgroups could initially experience a less effective model, such as a speech recognition function performing less accurately for women than for men. Therefore, it's crucial to identify subgroups where the model might exhibit poor performance, ideally gathering more data from these groups or, at the very least, giving greater consideration to these samples during model training and evaluation.

Concept drifts

Concept drifts happen when external changes or events occur that we did not record in our data or that change the meaning of our data. This means that the exact same input features suddenly result in a different output. One reason can be that we're **missing a variable that has a direct influence on the target**, for example

- our process is sensitive to temperature and humidity, but we only recorded the temperature not the humidity, so as the humidity changes, the same temperature values result in different output values additionally include humidity as an input feature in the model
- seasonal trends result in changes in the popularity of summer vs. winder clothes include month / outside temperature as an additional input feature
- special events, e.g., a celebrity mentioned our product on social media or people changed their behavior because of the lockdown during a pandemic while it can be hard to predict these events

in advance, when they happen we could include an additional feature, e.g., 'during lockdown', to distinguish data collected during this time period from the rest of the data
- degenerate feedback loops, i.e., the existence of the model changes users' behavior, e.g., a recommender system causes users to click on videos just because they were recommended include as an additional feature whether the video was recommended or not to learn how much of "user clicked on item" was due to the item being recommended and how much was due to the user's natural behavior

Another cause of concept drifts are events that **change the meaning of the recorded data**, for example

- inflation: 1 Euro in 1990 was worth more than 1 Euro now adjust the data for inflation or include the inflation rate as an additional input feature
- a temperature sensor immersed in water amasses limescale and after a while the temperature reading is not accurate anymore, e.g., if the true temperature is 90 degrees, a clean senor measures the true 90 degrees, but after it has accumulated some layers of limescale, it only measures 89 degrees under the same circumstances. While our output is influenced by the true temperature, we only have access to the sensor reading for the temperature, which is additionally influenced by the state of the sensor itself try to estimate the amount of accumulated limescale, e.g., based on the number of days since the sensor was cleaned the last time (which also means that these kinds of maintenance events need to be recorded somewhere!)

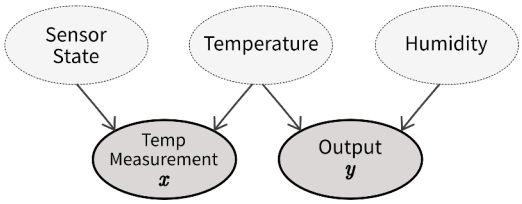


Figure 8: Causal diagram showing how our observed input x (temperature measurement) and output y are related through and influenced by hidden variables (which we can not access directly), namely, the state of the temperature sensor (i.e., how much limescale has accumulated), the actual temperature, and the humidity (for which we have not installed a sensor yet). If the sensor state and humidity stay constant, we are able to predict the output from the temperature measurement, however, if either of these values change, we experience a concept drift. Therefore, we should try to include estimates of these hidden variables in our model to account for these changes.

Before training a model, examine the data to **identify instances where identical inputs yield different outputs**. If possible, **include additional input features to account for these variations**. Subpar model performance on the test set often indicates missing relevant inputs, heightening vulnerability to future concept drifts. Even when the correct variables are incorporated to capture a concept drift, frequent model retraining may still be necessary. For instance, different states of the concept might be sampled unevenly, leading to data drifts (e.g., more data collected during winter than in the early summer months). If it is not possible to include variables that account for the concept drift, it might be necessary to **remove samples from the original training set that do not conform to the novel input/output relation before retraining the model**.

> **Tip**
>
> The best way to counteract data and concept drifts is to frequently **retrain the model** on new data. This can either happen on a schedule (e.g., every weekend, depending on how quickly the data changes) or when your monitoring system raises an alert because it detected drifts in the inputs or a deteriorating model performance.

Conclusion

Now that you've learned a lot about the machine learning (ML) theory, it is time for a reality check.

Hype vs. Reality

In the introduction, we've seen a lot of examples that contribute to the ML hype. However, especially when applying ML in the manufacturing industry, for example, the reality often looks quite different and not every idea might work out as hoped:

Hype: *Big Data, Generative AI*	Reality:
Database with millions of examples	150 manual entries in an excel sheet
Homogeneous unstructured data (e.g., pixels, sound, text)	Measurements from different sources with different scales (e.g., temperature, flow, pressure sensors)
Fancy deep learning architectures	Neural networks are tricky to train and even more difficult to explain → Need to understand and trust predictions to make business decisions

> **Note**
>
> But it can be done! A good example comes from the startup alcemy, which uses ML to optimize the production of CO2-reduced cement. They describe how they overcame the above mentioned challenges in this talk.

Machine Learning is just the tip of the iceberg

You were already warned that in their day-to-day operations, data scientists usually spend only about 10% of their time doing the fun machine learning stuff, while the bulk of their work consists of gathering and cleaning data. This is true for an individual ML project. If your goal is to become a data-driven enterprise that uses AI in production for a wide range of applications, there are some additional challenges that should be addressed – but which would typically not be the responsibility of a data scientist (alone):

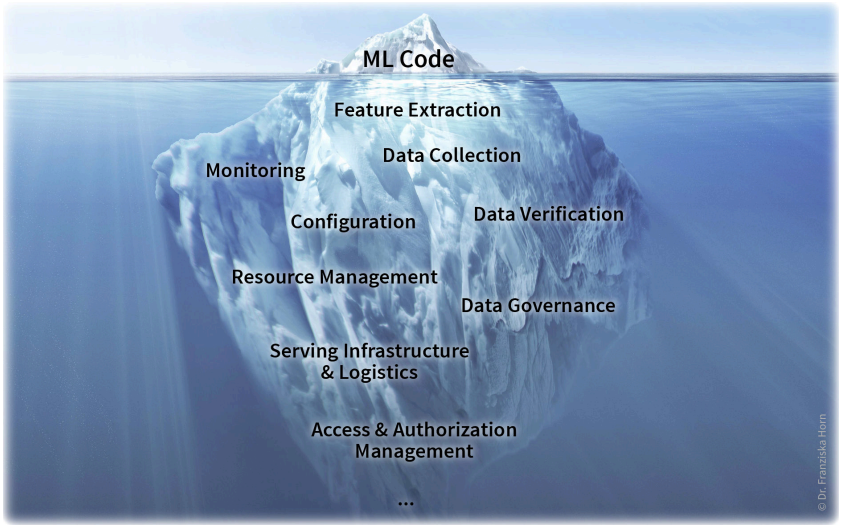
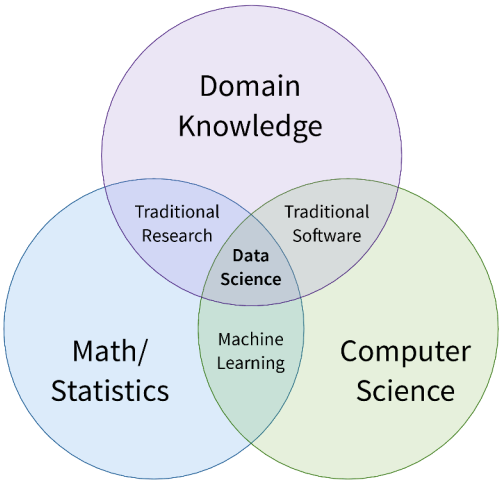


Figure 1: See also: Sculley, David, et al. "Hidden technical debt in machine learning systems." Advances in Neural Information Processing Systems. 2015.

On the plus side, things like a centralized data infrastructure and clear governance process only need to be set up once and then all future ML projects will benefit from them.

Domain knowledge is key!

In the introduction, you've seen the Venn diagram showing that ML lies at the intersection of math and computer science. However, this is actually not the complete picture. In the previous chapters, you've hopefully picked up on the fact that in order to build trustworthy models that use meaningful features to arrive at robust conclusions, it is necessary to combine ML with some domain knowledge and understanding of the business problems, what is then often referred to as Data Science:



As we will argue in the next section, it is unrealistic to expect an individual data scientist to be an expert in all three areas, and we therefore instead propose three data-related roles to divide responsibilities in an organization.

Take Home Messages

- ML is and will be transforming all areas of our lives incl. work.
- ML has limitations:
  - Performance: Some problems are hard.
  - Data Quality & Quantity: Garbage in, garbage out!
  - Causality & Adversarial Attacks Explainability!!
- Combine ML with subject matter expertise!
- It's an iterative process:
  - Don't expect ML to work right away!
  - Monitor and update after initial release.

If you've now gotten curious and want to know more about how the different machine learning algorithms actually work, have a look at the full version of this book!

AI Transformation of a Company

The famous ML researcher Andrew Ng has proposed a five-step process to transform your company into a data-driven enterprise capable of using AI in production to add value.

Five steps for a successful AI Transformation by Andrew Ng

1. Execute pilot projects to gain momentum
2. Build an in-house AI team & data infrastructure
3. Provide broad AI training (for all employees)
4. Develop an AI & data strategy
5. Develop internal and external communications

Recommended Materials:
→ "AI for everyone" Coursera course
→ AI Transformation Playbook

(a) **Prof. Dr. Andrew Ng**
Co-Founder Google Brain
Vice President Baidu
Co-Founder Coursera
Professor @ Stanford University

[Step 1] Start with small pilot projects to understand the potential and challenges of using ML

Machine learning projects are unlike traditional software projects, where you're usually certain that a solution at least exists and you only need to figure out an efficient way to get there. Instead, ML heavily relies on the available data. Even though it might theoretically be possible to solve your problem with ML, this might not be the case with the data you have at hand. Before implementing

some big AI initiative spanning the whole company, it is therefore strongly recommended that you start with several smaller pilot projects in order to get a better feeling for what it means to rely on an AI to solve your problems.

When choosing a pilot project, the most important factor is not the Return on Investment (ROI) of the project, since here the experience with ML gained along the way should be the priority. However, it is important to choose a project that is **technically feasible**, i.e., which can be solved with existing ML algorithms and you don't need years of research to develop your own fancy neural network architecture. Furthermore, you should have enough high-quality **data available** to get started, so you don't spend months just on data preprocessing, e.g., due to the need to combine data from different sources within a poor data infrastructure.
If you do not yet have the necessary AI talent in-house to tackle such a project, you can also partner with external consultants, which provide the ML expertise, while you supply the subject matter expertise to ensure the pilot project is a success.

[Step 2] Set up a centralized AI team and data infrastructure to carry out bigger projects efficiently and effectively

We've already seen that in practice, it's really about the intersection of Theory, Programming, and Domain Knowledge, i.e., Data Science. However, it is unlikely that you'll find a single person that is truly competent in all three areas. Instead, people will always have a certain focus and we therefore propose three distinct roles, which also align very well with the three main steps for successfully executing an ML project:

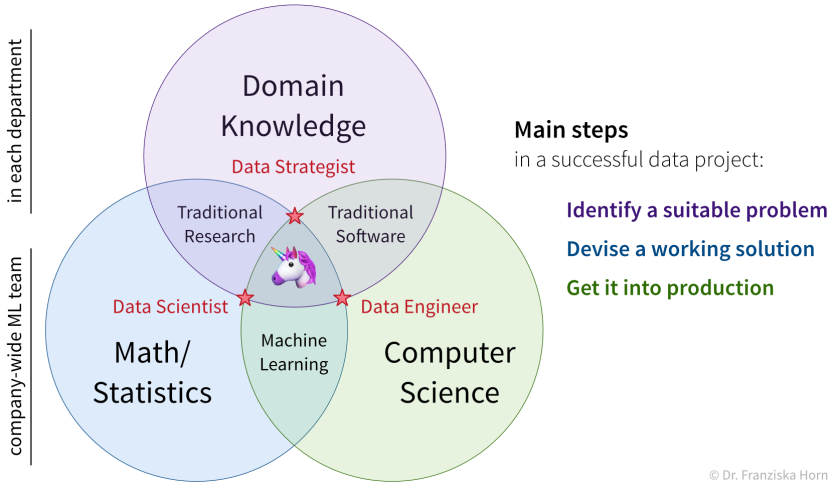


Figure 3: While Data Strategists work in their respective departments to identify suitable problems that can benefit from ML, Data Scientist can experiment and develop prototypical solutions to these problems, which Data & ML Engineers then get ready for production.

Ideally, data scientists and engineers should be in their own separate team (i.e., the "AI Team") and work on projects from different departments like an in-house consultancy:

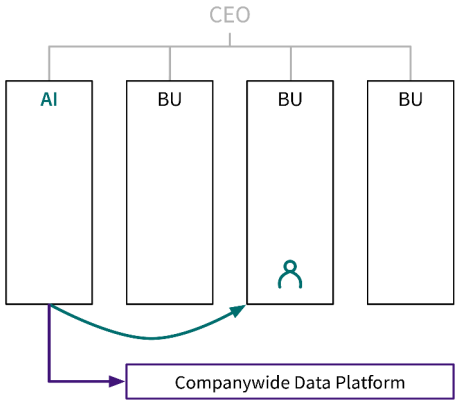


Figure 4: [Adapted from: "AI for everyone" by Andrew Ng (coursera.org)]

This has several advantages:

- Data scientists can discuss solutions with other ML experts → many problems will be similar from an algorithmic standpoint.
- Combine data from the whole company for a holistic analysis.
- Funding independent from individual business unit, e.g., necessary for the up front investment in data infrastructure, time required to keep up with new research, etc.

As we've discussed in the introduction, about 90% of the time in an ML project is spent on data wrangling. Therefore, especially in the beginning, **the AI team should contain more Data Engineers than Data Scientists**, so they can build a solid data infrastructure, which will save Data Scientists lots of time and headaches later.

[Step 3] Train other employees to recognize ML problems and establish a data-driven culture

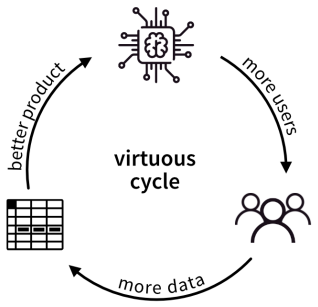
While data scientists need to be intimately familiar with the algorithms they are using, other employees, especially data strategists and department leaders, should have some basic understanding of what ML is and is not capable of, such that they can identify possible ML problems and refer them to the AI team.

Figure 5: I have devised trainings at different levels for all audiences.

[Step 4] Devise a cohesive strategy with long-term goals that result in a competitive advantage

Developing a strategy might be the first impulse of an executive when confronted with a new topic such as AI. However, since AI problems are so different from other kinds of projects, it really pays off to first gain some experience with this topic (i.e., start with step 1!). After you've successfully completed some pilot projects and set the wheels in motion to create an AI team as well as educate the other employees to get them on board, here are a few things to consider w.r.t. a companywide strategy to give you an advantage over your competition:

- Create strategic data assets that are hard for your competition to replicate:
  - Long-term planning: Which data might be valuable in the future? → Start collecting it now!
  - Up-front investments: What infrastructure and processes are needed to make the data accessible to the right people?
  - How can you combine data from different divisions to enable the AI team to "connect the dots" and gain a unique edge over the competition?
  - What options do you have in terms of strategic data acquisition, e.g., in the form of 'free' products, where users pay with their data (like what Google, Facebook, etc. are doing)?
- Build AI-powered features that are a unique selling point for your products:
  - Don't try to recreate some off-the-shelf service that could be easily procured from an outside vendor, but use ML together with your unique subject matter expertise and data to build new features for your existing products to make them more appealing to your customers or open up new market segments.
  - How can you establish a virtuous cycle, where your AI attracts more users, which in turn generate more data, which can then be used to train the AI to become even better and thereby attracts even more users?



[Step 5] Communicate your success

After successfully implementing AI within the company, you should of course communicate your accomplishments. In addition to internal and external press releases, this also includes, for example, job listings, which will attract more qualified candidates if they are formulated from an informed standpoint instead of listing buzzwords.