Andriy Burkov

# MACHINE LEARNING ENGINEERING

# Machine Learning Engineering

Andriy Burkov

ISBN 978-1-9995795-7-9

Publisher: Andriy Burkov

To my parents:
Tatiana and Valeriy

and to my family:
daughters Catherine and Eva,
and brother Dmitriy

# Contents

# Preface

There is plenty of good books on machine learning, both theoretical and hands-on. You can learn from a typical machine learning book the types of machine learning, major families of algorithms, how they work and how to build models from data using those algorithms.

A typical machine learning book is less concerned with the engineering aspects of the implementation of machine learning projects. Such questions as data collection, storage, preprocessing, feature engineering, as well as testing and debugging of models, their deployment to and retirement from production, runtime and post-production maintenance are often either completely left outside the scope of machine learning books or considered superficially.

This book fills that gap.

## Who This Book is For

In this book, I assume that the reader understands the machine learning basics and is capable of building a model given a properly formatted dataset using a favorite programming language or a machine learning library[1].

The target audience of the book is data analysts who lean towards a machine learning engineering role, machine learning engineers who want to bring more structure to their work, machine learning engineering students, as well as software architects who frequently deal with models provided by data analysts and machine learning engineers.

## How to Use This Book

This book is a comprehensive review of machine learning engineering best practices and design patterns. I recommend reading it from beginning to end. However, you can read chapters in any order as they cover distinct aspects of the machine learning project lifecycle and don't have direct dependencies between each other.

[1] If it's not the case for you, I recommend reading The Hundred-Page Machine Learning Book first.

## Should You Buy This Book?

Like its companion and precursor The Hundred-Page Machine Learning Book, this book is also distributed on the "read first, buy later" principle. I firmly believe that readers have to be able to read a book before paying for it, otherwise, they buy a pig in a poke.

The *read first, buy later* principle implies that you can freely download the book, read it and share it with your friends and colleagues. If you read and liked the book, or found it helpful or useful in your work, business or studies, then buy it.

Now you are all set. Enjoy your reading!

# Chapter 1

# Introduction

## 1.1 What is Machine Learning

Although I assume that a typical reader of this book knows the basics of machine learning, it's still important to start with definitions, so that we are sure that we have a common understanding of the terms we use throughout the book.

I will repeat the definitions I give in my previous The Hundred-Page Machine Learning Book, so if you have that book, you can skip this section.

Machine learning is a subfield of computer science that is concerned with building algorithms which, to be useful, rely on a collection of examples of some phenomenon. These examples can come from nature, be handcrafted by humans or generated by another algorithm.

Machine learning can also be defined as the process of solving a practical problem by 1) gathering a dataset, and 2) algorithmically building a statistical model based on that dataset. That statistical model is assumed to be used somehow to solve the practical problem.

To save keystrokes, I use the terms "learning" and "machine learning" interchangeably.

Learning can be supervised, semi-supervised, unsupervised and reinforcement.

### 1.1.1 Supervised Learning

In **supervised learning**[1], the **dataset** is the collection of **labeled examples** $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. Each element $\mathbf{x}_i$ among $N$ is called a **feature vector**. A feature vector is a vector in which each dimension $j = 1, \ldots, D$ contains a value that describes the example somehow. That value is called a **feature** and is denoted as $x^{(j)}$. For instance, if each example $\mathbf{x}$ in our

[1] If a term is **in bold**, that means that the term can be found in the index at the end of the book.

collection represents a person, then the first feature, $x^{(1)}$, could contain height in cm, the second feature, $x^{(2)}$, could contain weight in kg, $x^{(3)}$ could contain gender, and so on. For all examples in the dataset, the feature at position $j$ in the feature vector always contains the same kind of information. It means that if $x_i^{(2)}$ contains weight in kg in some example $\mathbf{x}_i$, then $x_k^{(2)}$ will also contain weight in kg in every example $\mathbf{x}_k$, $k = 1, \ldots, N$. The **label** $y_i$ can be either an element belonging to a finite set of **classes** $\{1, 2, \ldots, C\}$, or a real number, or a more complex structure, like a vector, a matrix, a tree, or a graph. Unless otherwise stated, in this book $y_i$ is either one of a finite set of classes or a real number[2]. You can see a class as a category to which an example belongs.

For instance, if your examples are email messages and your problem is spam detection, then you have two classes $\{spam, not\_spam\}$. In supervised learning, the problem of predicting a class is called **classification**, while the problem of predicting a real number is called **regression**. Another example of classification is when a doctor shows to the model the properties of a patient and the model returns the diagnosis.

The goal of a **supervised learning algorithm** is to use the dataset to produce a **model** that takes a feature vector $\mathbf{x}$ as input and outputs information that allows deducing the label for this feature vector. For instance, the model created using the dataset of people could take as input a feature vector describing a person and output a probability that the person has cancer.

Even if the model is typically a mathematical function, when thinking about what model does with the input it is convenient to think that the model "looks" at the values of some features in the input and, based on past experience with similar examples, outputs a value. That output value is a number or class "the most similar" to the targets seen in the past in the examples with similar values of features. It looks very simplistic, but the decision tree model works almost like that.

### 1.1.2 Unsupervised Learning

In **unsupervised learning**, the dataset is a collection of **unlabeled examples** $\{\mathbf{x}_i\}_{i=1}^N$. Again, $\mathbf{x}$ is a feature vector, and the goal of an **unsupervised learning algorithm** is to create a **model** that takes a feature vector $\mathbf{x}$ as input and either transforms it into another vector or into a value that can be used to solve a practical problem. For example, in **clustering**, the model returns the id of the cluster for each feature vector in the dataset. Clustering is useful for finding groups of similar objects in a large collection of objects, such as images or text documents. By using clustering, for example, the analyst can sample sufficiently representative yet small subset of unlabeled examples from a large collection of examples for labeling: a few examples are sampled from each cluster instead of sampling directly from the large collection and risking only sampling examples very similar to one another.

In **dimensionality reduction**, the output of the model is a feature vector that has fewer features than the input $\mathbf{x}$. For example, the scientist has a feature vector, which is too complex

[2] A real number is a quantity that can represent a distance along a line. Examples: $0$, $-256.34$, $1000$, $1000.2$.

to visualize (it has more than three dimensions). The dimensionality reduction model can transform this feature vector into a new feature vector that has only two or three dimensions and, therefore, this new example can be plotted on a graph.

In **outlier detection**, the output is a real number that indicates how $\mathbf{x}$ is different from a "typical" example in the dataset. Outlier detection is useful for solving network intrusion problem (by detecting abnormal network packets which are different from a typical packet in a "normal" traffic) or detect novelty, such as a document different from the existing documents in a collection.

### 1.1.3 Semi-Supervised Learning

In **semi-supervised learning**, the dataset contains both labeled and unlabeled examples. Usually, the quantity of unlabeled examples is much higher than the number of labeled examples. The goal of a **semi-supervised learning algorithm** is the same as the goal of the supervised learning algorithm. The hope here is that using many unlabeled examples can help the learning algorithm to find (we might say "produce" or "compute") a better model.

It could look counter-intuitive that learning could benefit from adding more unlabeled examples. It seems like we add more uncertainty to the problem. However, when you add unlabeled examples, you add more information about your problem: a larger sample reflects better the probability distribution the data we labeled came from. Theoretically, a learning algorithm should be able to leverage this additional information.

### 1.1.4 Reinforcement Learning

**Reinforcement learning** is a subfield of machine learning where the machine "lives" in an environment and is capable of perceiving the *state* of that environment as a vector of features. The machine can execute *actions* in every state. Different actions bring different *rewards* and could also move the machine to another state of the environment. The goal of a reinforcement learning algorithm is to learn a *policy*.

A policy is a function (similar to the model in supervised learning) that takes the feature vector of a state as input and outputs an optimal action to execute in that state. The action is optimal if it maximizes the *expected average reward*.

Reinforcement learning solves a particular kind of problem where decision making is sequential, and the goal is long-term, such as game playing, robotics, resource management, or logistics.

## 1.2 When to Use Machine Learning

Machine learning is a powerful tool for solving practical problems, however, like any tool, it has to be used in the right context. Trying to solve all problems using machine learning would be a mistake.

You should consider using machine learning in one of the following situations.

### 1.2.1 When the Problem Is Too Complex for Coding

In a situation where the problem is so complex or big that you cannot hope to write all the code to solve the problem and where a partial solution is viable and interesting, you can try to solve the problem with machine learning.

One example is spam detection: it's impossible to write the code that will implement such a logic that will effectively detect spam messages and let genuine messages reach the inbox. There are just too many factors to consider. For instance, if you program your spam filter to reject all messages from people who are not in your contacts, you risk losing messages from someone who has got your business card on a conference. If you make an exception for messages containing specific keywords related to your work, you will probably miss a message from your child's teacher, and so on.

If you still decide to directly program a solution to that complex problem, with time you will have in your programming code so many conditions and exceptions from them that maintaining that code will eventually become infeasible. In this situation, training a classifier on examples "spam"/"not_spam" seems logical and the only viable choice.

### 1.2.2 When the Problem Is Constantly Changing

Some problems may continuously change with time so that the programming code has to be regularly updated. This results in the frustration of software developers working on the problem, an increased chance of introducing errors, difficulties of combining "previous" and "new" logic, and significant overhead of testing and deploying updated solutions.

For example, you can have a problem of scraping specific data elements from a collection of webpages. Let's say that for each webpage in that collection you write a set of fixed data extraction rules in the following form: "pick the third <p> element from <body> and then pick the data from the second  inside that <p>". If the website owner changes the design of the webpage, the data you scrape may end up in the second or the fourth <p> element, making your extraction rule wrong. If the collection of webpages you scrape is large (thousands of URLs), some rules will become wrong all the time and you will endlessly fix those rules.

### 1.2.3 When It Is a Perceptive Problem

Today, it's hard to imagine someone trying to solve perceptive problems such as speech, image, and video recognition without using machine learning. Consider an image. It's represented by millions of pixels. Each pixel is given by three numbers: the intensity of red, green and blue channels. In the past, engineers tried to solve the problem of image recognition (detecting what's on the picture) by applying hand-crafted "filters" to square patches of pixels. If one filter, for example, the one that was designed to "detect" grass, generates a high value when applied to many pixel patches, while another filter, designed to detect brown fur, also returns high values for many patches, then we can say that there are high chances that the image represents a caw on the field (I'm simplifying a bit).

Today, perceptive problems are solved using machine learning techniques, such as neural networks.

### 1.2.4 When the Problem Has Too Many Parameters

Humans have a hard time with prediction problems based on input that has too many parameters or they are correlated in unknown ways. For example, take the problem of predicting whether the borrower will repay the loan. Each borrower is represented by hundreds of numbers: age, salary, account balance, frequency of past payments, married or not, amount of children, make and year of the car, mortgage balance, and so on. Some of those numbers may be important to make the decision, some may be less important alone, but become more important if considered in combination with other numbers. Writing a code that will make such decisions is hard because even for a human it's not clear how to combine all those numbers describing a person in an optimal way into a prediction.

### 1.2.5 When It Is an Unstudied Phenomenon

If we need to make predictions of some phenomenon, which is not well studied scientifically but examples of it are observable, then machine learning might be an appropriate (and in some cases the only available) option. For example, machine learning can be used to generate personalized mental health medication options based on genetic and sensory data of a patient. Doctors might not necessarily be able to interpret such data to make an optimal recommendation, while a machine can discover patterns in data by analyzing thousands of training examples and predict which molecule has highest chances to help a given patient.

Another example of observable but unstudied phenomena are logs of a complex computing system or a network. Such logs are generated by multiple independent or interdependent processes. For a human, it's hard to make predictions about the future state of the system based on logs alone without having a model of each process and their interdependency. If the amount of examples of historical logs is high enough (which is often the case) the machine

can learn patterns hidden in logs and be able to make predictions without knowing anything about each individual process.

Finally, making predictions about people based on their observed behavior is hard. In this problem, we obviously cannot have a model of a person's brain, but we have easily available examples of expression of the person's ideas (in form of online posts, comments, and other activities). Based on these expressions alone, a machine learning model deployed in a social network can recommend the content or other people to connect with, without having a model of the person's brain.

### 1.2.6 When the Problem Has a Simple Objective

Machine learning is especially suitable for solving problems which you can formulate as a problem with a simple objective: such as yes/no decisions or a single number. In contrast, you cannot use machine learning build a model that works as an operating system because there are too many different decisions to make; getting examples that illustrate all (or even most) of those decisions is practically infeasible.

### 1.2.7 When It Is Cost-Effective

Three major sources of cost in machine learning are:

- building the model,
- building and running the infrastructure to serve the model,
- building and running the infrastructure to maintain the model.

The cost of building the model includes the cost of gathering and preparing data for machine learning. Model maintenance includes continuously monitoring the model and gathering additional data to keep the model up to date.

## 1.3 When Not to Use Machine Learning

There are plenty of problems which cannot be solved using machine learning, it's hard to characterize all of them. Here I will give several hints. You probably should not use machine learning when:

- any action of the system or a decision made by it has to be explainable,
- any change in the system's behavior has to be explainable,
- the cost of making an error is too high,
- it's crucial to get to the market first,

- getting right data is too complex or impossible,
- you know how to solve the problem using traditional software development and the cost will be lower,
- the phenomenon has too many outcomes while you cannot get enough of examples to represent those outcomes (like in our operating system example),
- you build a system for which you are sure that it will not have to be improved frequently over time,
- you can fill an exhaustive lookup input-output database table manually (the number of possible input values is not too large).

## 1.4 What is Machine Learning Engineering

Machine learning engineering (MLE) is the use of scientific principles, tools, and techniques of machine learning and traditional software engineering to design and build complex computing systems. MLE encompasses all stages from data gathering, to model building, to making the model available for use by the product or the consumers.

Typically, a data analyst is concerned with understanding the business problem, building a model to solve that problem and evaluating it in a restricted development environment. A machine learning engineer, in turn, is concerned with sourcing the data from various sources, preprocessing it, programming features, building an efficient model, which will run in the production environment, coexist well with other production processes, be stable, maintainable and easily accessible by different types of users with different use cases.

In other words, MLE concerns every process that allows machine learning algorithms to be implemented as part of an effective production system.

In practice, machine learning engineers might be employed in such activities as rewriting a data analyst's code from rather slow R and Python into more efficient Java or C++, scaling this code to make it more robust, packaging the code into an easy-to-deploy versioned package, optimizing the machine learning algorithm to make sure it is compatible with and runs properly in the company's production environment. In many companies, data analysts execute some of the MLE tasks, such as data extraction, transformation, or feature engineering. In some companies, machine learning engineers execute some of the data analysis tasks, including learning algorithm selection, hyperparameter tuning, and model evaluation.

A machine learning model needs more attention than a typical computer program. For example, during the first weeks after the deployment in production, it can work perfectly fine and then start returning inconsistent or completely wrong results. Such behavior of the model might be explained by a fundamental change in the input data or an updated feature extractor that now returns a different distribution of values or one of many other reasons we will consider in this book. They often say that machine learning systems *fail silently*. The machine learning engineer must be capable of preventing such failures or, when it's impossible to prevent them, know how to detect and handle them when they happen.

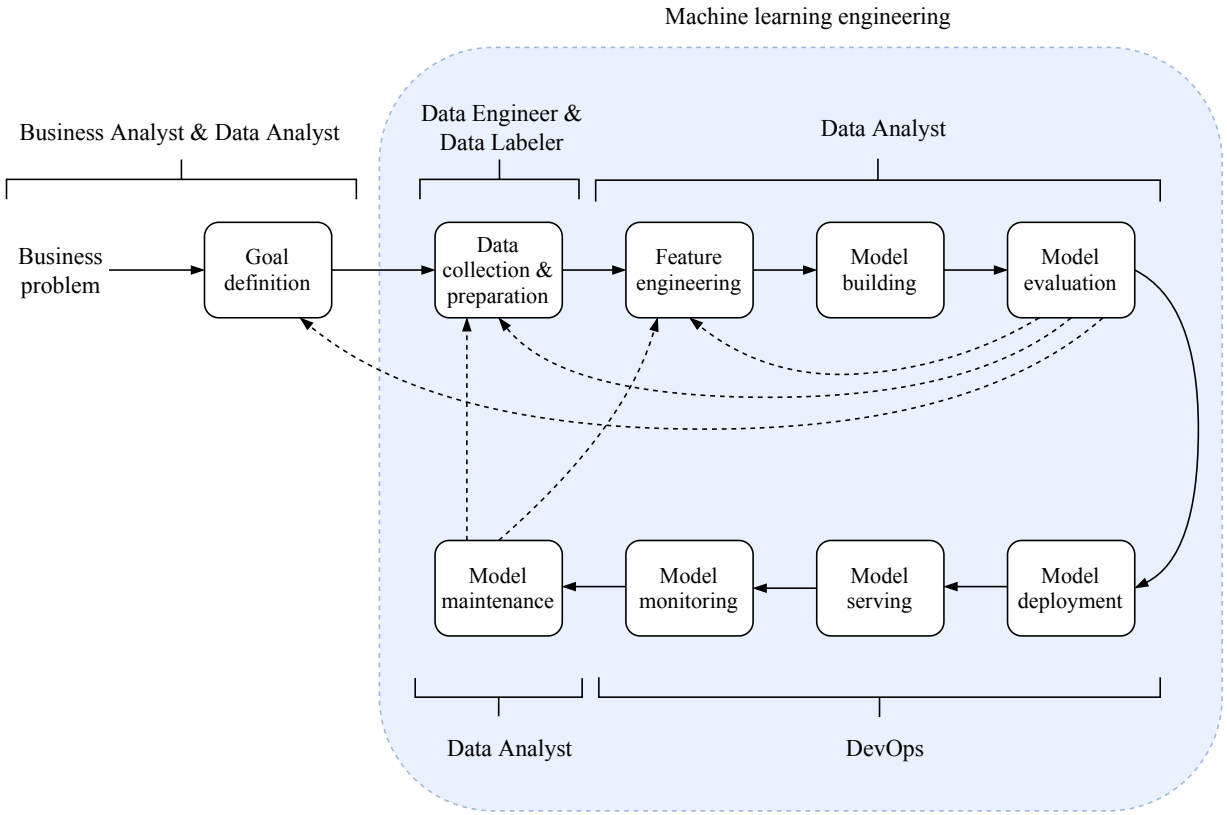


Figure 1.1: Machine learning project life cycle.

## 1.5 Machine Learning Project Life Cycle

Machine learning project starts with understanding the business objective. Usually, a business analyst works with the client[3] and the data analyst to transform a business problem into an engineering project. The engineering project may or may not have a machine learning part. In this book, we, of course, consider engineering projects that have *some* machine learning involved.

Once an engineering project is defined, this is where the scope of the machine learning engineering starts. In the scope of the broader engineering project, machine learning must first of all have a well defined **goal**. The goal of machine learning is a specification of what a

[3] If the machine learning project supports a product developed and sold by the organization, then the business analyst works with the product owner.

statistical model receives as input, what it generates as output, and the criteria of acceptable (or unacceptable) behavior of the model.

The goal of machine learning is not necessarily the same as the business objective. The business objective is what the organization wants to achieve. For example, the business objective of Google with Gmail can be to make Gmail the most used email service. Google might create multiple machine learning engineering projects to achieve that business objective. The goal of one of those machine learning projects can be to distinguish personal emails from promotional with accuracy above 90%.

Overall, a machine learning project life cycle, illustrated in Figure 1.1 consists of the following stages:

- Goal definition
- Data gathering
- Feature engineering
- Model building
- Model evaluation
- Model deployment
- Model serving
- Model monitoring
- Model maintenance

In Figure 1.1, the scope of machine learning engineering (and the scope of this book) is shown by the blue zone. The solid arrows show a typical flow of the project stages. The dashed arrows show that at some stages, a decision to can be made to go back in the process and either gather more data or gather different data and revise features (by decommissioning some of them and engineering new ones).

For each of the above stages, there's a distinct chapter in the book. But first of all, let's discuss how to prioritize machine learning projects, how to define the project's goal, and how to structure the machine learning team. The next chapter is devoted to these three questions.

# Chapter 2

# Before the Project Starts

Before a machine learning project starts, it has to be prioritized and the team working on the project has to be built. Prioritization is unavoidable, because the team and equipment capacity is typically limited, while the backlog list of projects a company has might be very long.

## 2.1 Prioritization of Machine Learning Projects

The key considerations in the prioritization of machine learning project are *impact* and *cost*.

The impact of using machine learning in a broader engineering project is high when 1) machine learning can replace a complex part in your engineering project or 2) there's a high benefit in getting inexpensive (and probably noisy) predictions.

For example, some complex part of an existing system can be rule-based, with many rules, rules inside rules, and exceptions from rules. Building and maintaining such a system can be extremely difficult, time-consuming and error-prone. It can also be a source of major frustration for software engineers to work on maintaining that part of the system. Can the rules be learned instead of programming them? Can the existing system be used to generate annotated data easily? If yes, such a machine learning project would have a high impact and probably low cost.

Noisy inexpensive predictions can be valuable, for example, in a system that dispatches a big amount of requests. Let's say many such requests are "easy" and can be solved quickly using some existing automation. The remaining requests are considered "difficult" and must be addressed manually. A machine learning-based system that recognizes "easy" tasks and dispatches them to the automation will save a lot of time for humans who will only concentrate their effort and time on difficult requests. Even if the dispatcher makes an error

in prediction, the complex request will reach the automation, the automation will fail on it, and the human will eventually receive that request. If the human receives a simple request by mistake, it's also not a problem: that simple request can still be sent to the automation or processed by the human (to avoid the overhead of forwarding the request to automation).

The cost of the machine learning project is highly influenced by three factors:

- the difficulty of the problem,
- the cost of data and
- the needed accuracy.

Getting the right data and the right amount of it can be very costly, especially if a manual annotation is needed. The need for high accuracy can translate into the requirement of getting more data or building a more complex model, such as a unique architecture of a deep neural network or a nontrivial ensembling architecture.

When you think about the problem's difficulty, the main considerations are:

- whether an implemented algorithm or a software library capable of solving the problem is available (if yes, the problem is greatly simplified),
- whether a significant compute is needed to build the statistical model,
- whether a significant compute is needed to run the model in the production environment.

The second driver of the cost is data. The following considerations have to be made:

- can data be generated automatically (if yes, the problem is greatly simplified),
- what is the cost of manual annotation of the data (i.e., assigning labels to unlabeled examples),
- how many examples are needed (usually, this cannot be known in advance, but can be estimated from known published results or company's own past experience).

Finally, one of the most important factors influencing the cost is the desired accuracy of the model. The consideration to be aware of is that the cost of the machine learning project grows superlinearly in the accuracy requirement as illustrated in Figure 2.1. Low accuracy can also be a source of significant loss in the future when the model will be deployed in the production environment. The considerations to make:

- How costly is each wrong prediction?
- What is the lowest accuracy level below which the model becomes too noisy for being practical?

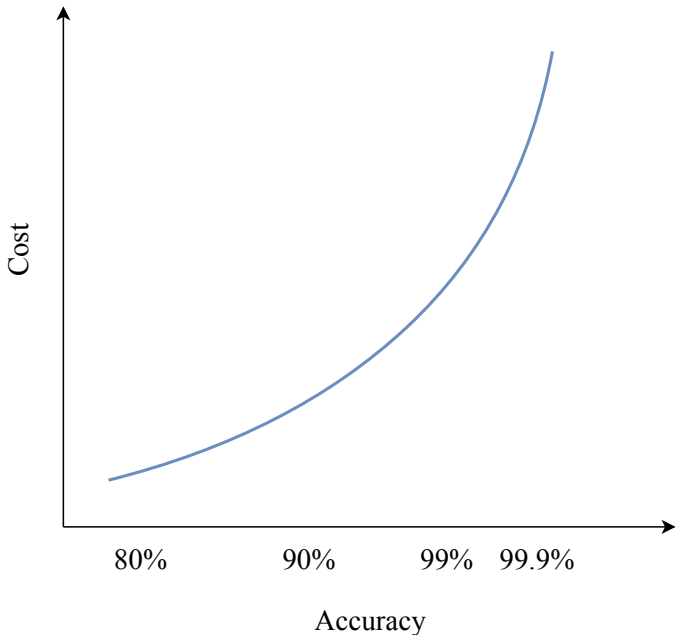


Figure 2.1: Superlinear growth of the cost as a function of accuracy requirement.

## 2.2 Estimating Complexity of a Machine Learning Project

There is no standard way to estimate how complex a machine learning project is other than by comparison with other projects executed by the company or reported in the literature. There are several major unknowns that are almost impossible to guess with confidence unless you worked on a similar project in the past or read about such a project. The unknowns are:

- whether the required accuracy level (or the value of any other metrics important to you) is attainable in practice,
- how much data you will need to reach the required accuracy level,
- how many features (and which ones) are needed so that the model can learn and generalize sufficiently well,
- how large the model has to be (especially relevant for neural networks and ensemble architectures),
- how long will it take to train one model (so-called *experiment*) and how many experiments will be needed to reach the desired level of performance.

One thing you can almost be sure of is that if the required accuracy level is above 99% you can expect complications related to an insufficient quantity of labeled data. In some problems, even 95% accuracy is considered very hard to reach.

Another useful reference is the human performance on the task. If you want your model to perform as well as a human, this is typically a hard problem.

One way to make a more educated guess is to simplify the problem and try to solve it first. For example, you have a problem of classifying a set of documents into 1000 topics. Run a pilot project by focusing on 10 topics first, by considering documents belonging to other 990 topics as "Other". Annotate the data for these 11 classes (10 real topic plus Other). The logic here is that it's much simpler for a human to keep in mind the definitions of only 10 topics compared to memorizing the difference between 1000 topics. To save even more time, apply clustering to the whole collection of unlabeled documents and only label documents belonging to one or a few clusters.

Then solve the problem for 11 classes and measure time on every stage. Once you see that the problem for 11 classes is solvable you can hope that it will be solvable for 1000 classes as well. Your measurements can then be used to estimate the time required to solve the full problem, though you cannot multiply this time by 100 to get an accurate estimate. The quantity of data needed to learn to distinguish between more classes usually grows superlinearly with the number of classes.

An alternative way of obtaining a simpler problem from a potentially complex one is to split the problem using the slices in the available data. For example, you want to predict something about your customers and you have customers in multiple locations. Solve the problem for one location only, or for customers in a specific age range.

The progress of machine learning project is nonlinear. The accuracy usually grows fast in the beginning, but then the growth slows down. Sometimes you see no growth and decide to add additional features potentially depending on external databases or knowledge bases. While you work on a new feature or annotate more data (or outsource this task to an external vendor or team), no progress in the level of accuracy is happening at all.

Because of this non-linearity in progress, you have to make sure that the product leader (or the client) understands the constraints and the risks. Carefully log every activity and track the time it took. This will help not only in reporting but also in simplifying complexity estimations for similar projects in future.

## 2.3 Defining the Goal of a Machine Learning Project

The **goal** of a machine learning project is to build a model. The model is specified by the structure of its input (or inputs) and output (or outputs) and the minimum acceptable level of performance (as measured by accuracy of prediction or another performance metric).

The model will then be used as a part of a system that serves some purpose. In particular, the model can be used in a broader system to:

- **automate** (for example by taking an action on the user's behalf or by starting or stopping a certain activity on a server),

- **alert** or **prompt** (for example by asking the user if an action should be taken or a system administrator if the traffic seems suspicious),
- **organize** by presenting a set of items in an order that might be useful for a user (for example by sorting pictures or documents in the order of similarity to a query or according to user's preferences),
- **annotate** (for example, by adding contextual information to a display or by highlighting relevant phrases to user's task),
- **extract** (for example, by detecting smaller pieces of relevant information in a larger input, such as named entities in the text: proper names, companies, and locations),
- **recommend** (for example, by detecting and showing to a user highly relevant items in large collection based on item's content or user's reaction to past recommendations),
- **classify** (for example, by dispatching input examples into one, or several, of a predefined set of distinct named groups),
- **predict** (for example, by assigning a number, such as a price, to an object, such as a house),
- **synthesize** (for example, text, image, sound or a new object similar to other objects in a collection of objects),
- **answer an explicit question** (for example, "Does this text describe this image?" or "Are these two images similar?"),
- **simplify the input** (for example, by reducing its dimensionality for visualization purposes),
- **detect novelty** or an **anomaly**.

Almost any business problem solvable with machine learning can be defined in a form similar to one from the above list. If you cannot define your business problem in such a form, it's likely that machine learning is not the best solution in your case.

A successful model has the following four properties:

- it respects input and output specification and the minimum performance requirement,
- it benefits the organization (measured via sales and profit),
- it benefits the user (measured via productivity, engagement, and sentiment),
- it's scientifically rigorous.

A scientifically rigorous model is characterised by a *predicatble behavior*, for inputs that are from an operating range, and is *reproducible*. The former property means that if input feature vectors come from the same range of values as the training data, then the model, on average, has to make the same amount of errors as was observed on the holdout data (i.e. the data not used for training the model, but coming from the same statistical distribution as the training data) when the model was built. The latter property means that a model with similar characteristics can be *easily* built once again from the same training data using the same algorithm and values of hyperparameters. The word *easily* means that no additional analysis work is necessary to rebuild the model, only the compute power.

When defining the goal of machine learning, make sure you solve the right problem. To give an example of the incorrectly defined goal, imagine that your client has a cat and a dog and needs a system that lets their cat in the house but keeps their dog out. You might decide to train the model to distinguish cats from dogs. However, this model will also let *any cat* in and not just *their* cat. Alternatively, you may decide that because the client only has two animals, you will train a model that distinguishes between those two. In that latter case, because your classification model is binary, the raccoon will be classified as either a dog or a cat. If it's classified as a cat, it will be let in the house[1].

## 2.4 Structuring a Machine Learning Team

There are two cultures of structuring a machine learning team depending on the company.

The first culture says that a machine learning team has to be composed of data analysts (or scientists) who collaborate closely with software engineers. In such a culture, the software engineer doesn't need to have very deep expertise in machine learning but has to understand the vocabulary of their fellow data analysts or scientists.

According to the second culture, all engineers in a machine learning team must have a combination of machine learning and software engineering skills.

There are pros and cons in each culture. The proponents of the former say that each member of the team has to be the best in what they do. A data analyst must be an expert in many machine learning techniques and have a deep understanding of the theory to come up with an effective solution to most problems fast and with the least effort. Similarly, a software engineer has to have a deep understanding of various computing frameworks and be capable of writing efficient and maintainable code.

The proponents of the latter say that scientists are hard to integrate with software engineering teams. Scientists care more about how accurate their solution is and often come up with solutions that are impractical and cannot be efficiently executed in the production environment. Also, the fact that scientists usually don't write efficient and well-structured code, it has to be rewritten into the production code by a software engineer, which, depending on the project, can turn out to be a daunting task.

Besides machine learning and software engineering skills, a machine learning team may include experts in data engineering (or data engineers) and experts in data labeling.

Data engineers are software engineers responsible for ETL (for Extract, Transform, Load). These three conceptual steps are part of a typical data pipeline. Data engineers use ETL techniques and create an automated pipeline in which raw data is transformed to analysis-ready data. Data engineers design how data must be structured and integrated from various resources. They write on-demand queries on that data or wrap the most frequent queries into

[1] This is why to have the class "Other" in your classification problem is almost always a good idea.

fast APIs to make sure that the data is easily accessible by analysts and other data consumers. Generally, data engineers are not expected to know any machine learning.

In most big companies data engineers work separately from machine learning engineers in a data engineering team.

Experts in data labeling are responsible for three activities:

- manually or semi-automatically assign labels to unlabeled examples according to the specification provided by machine learning engineers,
- build labeling tools,
- manage outsourced labelers.

Again, in big companies, data labeling experts are organized in two or three different teams: one or two teams for data labelers (for example, one local and one outsourced) and a team of software engineers and a UX specialist responsible for building labeling tools.

Finally, DevOps engineers work closely with machine learning engineers to automate model deployment, loading, monitoring, and occasional or regular model maintenance. In smaller companies and startups, a DevOps engineer may be part of the machine learning team, or a machine learning engineer could be responsible for the DevOps activities. In big companies, DevOps engineers employed in machine learning projects usually work in a larger DevOps team.

# Chapter 3

# Data Collection and Preparation

Before any machine learning activity can start, the right data has to be collected and prepared for machine learning. The data available to the analyst is not always "right" and is not always in a form that a machine learning algorithm can use. In this chapter, I focus on types and forms of data, the properties of good quality data and typical problems a dataset can have.

## 3.1 Data Used Directly and Indirectly

Now that you have a machine learning goal with well-defined model input, output and success criteria, you can start collecting the data needed to train your model.

The data you need can be used to form the input examples *directly* or *indirectly*. For example, let's assume that we build a named entity extraction (NER) system. The input of the model is a sequence of words; the output is the sequence of labels[1] of the same length as the input. To make the data readable by a machine learning algorithm, we have to transform each natural language word into a machine-readable array of attributes, called **feature vector**.

Some features in the feature vector may contain the information that distinguishes that specific word from other words in the dictionary. Other features can contain additional attributes of the word, such as its shape (lowercase, uppercase, capitalized, and so on), and binary attributes indicating whether this word is the first word of some proper name or the name of some location or organization. To create these three latter binary features, we may decide to use some dictionaries, lookup tables or gazetteers.

You could already notice that the collection of word sequences is the data used to form training examples *directly*, while the data contained in dictionaries, lookup tables, and gazetteers is

[1] Labels can be, for example, values from the set {"Location", "Organization", "Person", "Other"}.

used *indirectly*: we can use it to extend feature vectors with additional properties but we cannot use it to create feature vectors.

## 3.2 Raw and Tidy Data

As we just discussed, directly used data is a collection of entities which is used as the basis to form a **dataset**. Each entity in that collection can be transformed into a training example, labeled or not. Raw data is a collection of entities in their natural form; such entities are not necessarily usable for machine learning. For example, a text document or an image are pieces of raw data; they cannot be directly used by a machine learning algorithm.

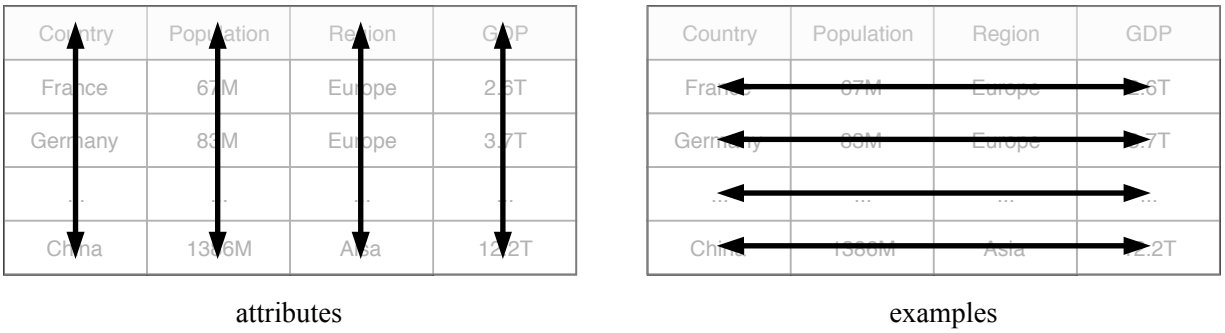


Figure 3.1: Tidy data: examples are rows and features are columns.

To be usable in machine learning a necessary (but not sufficient) requirement for the data is to be **tidy**. A tidy data can be seen as a spreadsheet, in which each row represents one example, and columns represent various **attributes** of an example, as shown in Figure 3.1. Sometimes raw data can be tidy, e.g. provided in the form of a spreadsheet. However, in practice to obtain tidy data from raw data, data analysts often use the procedure called **feature engineering**, which is applied to the direct and optionally indirect data with the goal of transforming each raw example into a feature vector, labeled or unlabeled[2].

It's important to highlight that data can be tidy, but still not usable by a certain machine learning algorithm. Most machine learning algorithms, in fact, only accept training data in the form of a collection *numerical* feature vectors. Consider the data in Figure 3.1. The attribute "Region" is *categorical* and not numerical. The decision tree learning algorithm can work with categorical values of attributes, but most learning algorithms cannot. In the next chapter, I will show how to transform a categorical attribute into a numerical feature[ˆch3_3].

[2]For some tasks, an example used by a learning algorithm can have a form of a sequence of vectors, a matrix, or a sequence of matrices. The notion of data tidiness for such algorithms is defined similarly. You only replace "row of fixed width in a spreadsheet" by a matrix of fixed width and height or a generalization of matrices to a higher dimension called a **tensor**.

In machine learning literature, when we use the word **example**, we typically mean a tidy data example with optionally assigned label to it. However, during the stage of data collection and labeling, examples can still be in the raw form: images, texts, or rows in a spreadsheet. In this book, when it's important to highlight the difference, I will say **raw example** to indicate that a piece of data was not transformed into a feature vector yet. Otherwise, assume that examples have the form of feature vectors.

The terms "attribute" and "feature" are often used interchangeably in machine learning literature. In this book, I use the term "attribute" to describe a specific property of an example, while the term "feature" refers to value $x^{(j)}$ at position $j$ in the feature vector $\mathbf{x}$ used by a machine learning algorithm.

We consider feature engineering in more detail in the next chapter.

## 3.3 Three Sets

In practice data analysts work with three distinct sets of examples:

1) training set,
2) validation set, and
3) test set.

Once you have got your directly used data in form of a collection of examples, tidy or not, the first thing you do is you shuffle the examples and split the dataset into three subsets: **training**, **validation**, and **test**. The training set is usually the biggest one; you use it to build the model. The validation and test sets are roughly the same sizes, much smaller than the size of the training set. The learning algorithm is not allowed to use examples from these two subsets to build the model. That is why those two sets are often called **holdout sets**.

There's no optimal proportion to split the data into these three subsets. In the past, the rule of thumb was to use 70% of the data for training, 15% for validation and 15% for testing. However, in the age of big data, datasets often have millions of examples. In such cases, it could be reasonable to keep 95% for training and 2.5%/2.5% for validation/testing.

The reason to have three sets and not one is simple: when we build a model, what we do not want is for the model to only do well at predicting labels of examples the learning algorithm has already seen. A trivial algorithm that simply memorizes all training examples and then uses the memory to "predict" their labels will make no mistakes when asked to predict the labels of the training examples, but such an algorithm would be useless in practice. What we really want is a model that is good at predicting examples that the learning algorithm didn't see: we want good performance on a holdout set.

We need two holdout sets and not one because we use the validation set to 1) choose the learning algorithm and 2) find the best values of **hyperparameters**. We use the test set to assess the model before delivering it to the client or putting it in production.

## 3.4 Questions About the Data

Before you start collecting the data, there are some questions to answer first.

### 3.4.1 Is the Data Accessible?

Does the data you need already exist? If yes, is it accessible (physically, contractually or from the cost perspective)? If it's accessible, is it protected by copyright or other legal norms? Is the data sensible (concerns your organization's projects, clients or partners, or is classified by the government) and are there any potential privacy issues to care about?

Even if it's physically possible to get the data you need, don't work with it until all other questions are resolved.

### 3.4.2 Is the Data Sizeable?

The question for which you would like to have a definitive answer is whether there's enough data. However, as I already mentioned, it's usually not known in advance how much data is needed to reach your goal, especially if the minimum accuracy requirement is very strict.

If you have doubts about the immediate availability of the sufficient amount of data, find out how frequently the new data gets generated. For some projects, you can start with a small initially available dataset and, while you work on feature engineering, modeling, and solving other relevant technical problems, the new data might gradually come in. The new data can come in either naturally, as a result of some observable or measurable process, or gradually provided by a third-party data provider.

Consider the estimated time needed to accomplish the project. Will a sufficiently large dataset be gathered during this time? Base your answer on the past experience working on similar projects or results reported in the literature.

One practical way to find out if you have collected sufficient data is to plot **learning curves**. More specifically, plot the training and validation scores of your learning algorithm for varying numbers of training examples as shown in Figure 3.2.

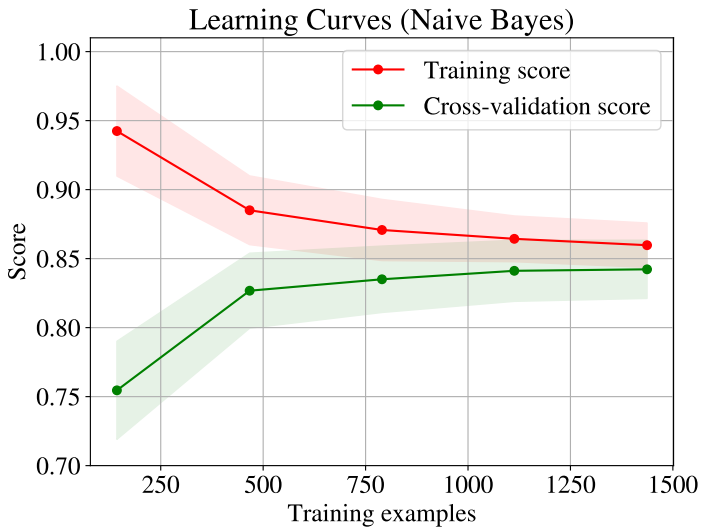


Figure 3.2: Learning curves for the Naïva Bayes learning algorithm applied to the standard *digits* dataset of scikit-learn.

By looking at the learning curves, you will see that the performance of your model will plateau after you reach a certain number of training examples. After reaching that number of training examples, you will begin to experience diminishing returns from adding each additional example.

If you observe that the performance of the learning algorithm plateaued, it *might* be a sign that collecting more data will not help in building a better model. Why I say "might be"? Because two other explanations are possible:

- You don't have enough informative features that your learning algorithm can leverage to build a more complex model, or
- You use a learning algorithm incapable of building a complex enough model using the data you have.

In the former case, you might think about engineering additional features by combining the existing features in some clever ways or by using information from indirect data sources, such as lookup tables and gazetteers.

In the latter case, you might think about using an ensemble learning method or training a deep neural network, though neural networks usually require more training data compared to shallow learning algorithms.

*You might not need all your big data*. In fact, just because you have big data does not mean that you should use all of it. A smaller sample of big data can give good results in practice

and accelerate the search for a better model. It's important to assure that the *sample is representative* of the whole big dataset. Such sampling strategies as stratified and balanced sampling can lead to better results. We consider data sampling strategies later in this chapter.

### 3.4.3 Is the Data Useable?

The quality of the data is one of the major factors affecting the performance of the model. Imagine that you want to build a model that predicts a person's gender given the name of that person. You might have a budget to acquire a dataset of people with gender information. If you blindly use this dataset you might realize later that no matter how hard you try to improve the performance of your model, its accuracy on new data is low. What is the reason for such a low performance? The answer could be that in the dataset you acquired the gender information was not factual but obtained using a statistical classifier of rather low quality. In this case, the best you can achieve with your own model is the level of performance of that low-quality classifier.

If the dataset comes in the form of a spreadsheet, the first thing to check is if the data in the spreadsheet is tidy. The dataset usable for machine learning has to be tidy. If it's not the case for your data, you will have to transform it into tidy data using feature engineering.

A tidy raw dataset can have **missing values**. If it's the case for your data, you can consider **data imputation** techniques to fill the missing values. We will consider several such techniques later in this chapter.

One frequent problem with dataset filled by humans is that people can decide to indicate missing values with some **magic number** like $9999$ or $-1$. During visual data analysis, such situations have to be spotted and these magic numbers have to be replaced using an appropriate data imputation technique.

Another property to validate is whether the dataset contains **duplicates**. Usually, duplicates are removed, unless you added them on purpose to balance an **imbalanced problem**. We consider this problem and methods to alleviate it later in this chapter.

Finally, data can be **expired** or significantly not up to date. For example, your goal is to build a model that recognizes abnormality when a complex piece of electronic appliance, such as a printer, misbehaves. You have measurements taken during normal and abnormal functioning of a printer. However, these measurements have been recorded for a previous generation of printers, while the new generation since then has received several significant upgrades. The model built using such outdated data coming from an older printer generation might perform worse when deployed on the new generation of printers.

Data can be **incomplete** or **unrepresentative** of the phenomenon of the study. A dataset of pictures of various animals can only contain pictures taken during the summer. A dataset of pedestrians for the self-driving car systems could be created by engineers with engineers as pedestrians; in such a dataset most situations on the road would involve younger men, while children, women and elderly people would be underrepresented or not present at

all. A company working on a facial expression recognition model could have the research and development office in a predominantly white location, so the dataset they would build might contain many faces of white men and women, while black or Asian people would be underrepresented. Another company working on posture recognition model to be deployed in a camera could take pictures of people indoors, while the users of the camera would use it predominantly outdoors.

In many practical cases, data can only become useable for modeling after preprocessing. Whence the importance of visual analysis of the dataset before you start modeling. Let's say you work on a problem of predicting the topic in the news articles. It's likely that you will get your data by scraping news websites. It's also likely that the name of authors of articles and the date when the text was downloaded would be saved together with the main text. Imagine also that the data engineer responsible for scraping decided to loop over news topics mentioned on the news websites and scrape news one topic at a time. So, on Monday the arts-related articles were scraped, on Tuesday - sports, on Wednesday - technology, and so on. It is also likely that names of article authors present in the text reveal the topic of the article because typically the same person writes on one or a few topics.

If you don't preprocess such data by removing the download dates and author names, the model can learn the correlation between specific dates or names present in the text and topics. Of course, such a model will be of no use in practice.

### 3.4.4 Is the Data Understandable?

As I already demonstrated above on the example of gender prediction, it is very important to understand where each attribute in the dataset came from. It is equally important to understand what exactly each attribute represents. One frequent problem observed in practice is when the variable which the analyst tries to predict is found among the features in the feature vector. How can this happen?

Imagine that you work on the problem of predicting the price of a house from its attributes such as the number of bedrooms, surface, location, construction year and so on. The attributes of each house are provided to you by the client, which is a big online real estate sales platform. The data has the form of an Excel spreadsheet. Without spending too much time on analyzing each column in the spreadsheet, you remove the transaction price from the dataset and use that value as the target that you want to model. Very quickly you realize that the model is almost perfect: it predicts the transaction price with the accuracy near 100%. You deliver the model to the client, they deploy it in production and the tests show that the model is wrong most of the time. What happened?

What happened is called the **data leakage** (also known as the **target leakage**). After a careful investigation of the dataset, you realize that one of the columns in the spreadsheet contained the real estate agent commission. Of course, the model easily learned to perfectly convert this attribute into the house price. However, this information is not available in the

production environment before the house is sold. Below, we will consider the problem of data leakage in more detail.

### 3.4.5 Is the Data Reliable?

The reliability of a dataset varies depending on the procedure that was used to gather that dataset. Can you trust the labels? If the data was produced by the workers on Mechanical Turk (so-called "turkers") then the reliability of such data can be very low. In some cases, the labels assigned by turkers to feature vectors are obtained as a majority vote of several turkers. If it's the case, then the data can be considered more reliable. However, even in this case, additional validation of quality has to be made for a small sample of the dataset.

On the other hand, if the data represents measurements made by some measuring devices, you can the details on the accuracy of each measurement in the technical documentation of the corresponding measuring device.

The reliability of labels is also affected by the **delayed** or **indirect** nature of the label. The label is considered delayed when the feature vector to which the label was assigned represents some event happened significantly earlier in time than the time of label observation. To be more concrete, take the **churn prediction** problem for example. In churn prediction, we have a feature vector describing a customer and we want to predict whether the customer will leave at some point in the future (typically 6 months to 1 year from now). The feature vector represents what we know about the customer *now*, but the label (customer left or stayed) will be obtained in the future. This an important property because between now and the future, many events, not reflected in our feature vector, can happen which can affect the customer's decision to stay or leave. Therefore delayed labels make our data less reliable.

Whether a label is direct or indirect also affect reliability, depending, of course, on what we are trying to predict. For example, let's say our goal is to predict whether the website visitor will be interested in a webpage. We might acquire a certain dataset containing information about users, webpages and labels "interested"/"not_interested" reflecting whether a specific user was interested in a specific webpage. A direct label would indeed indicate the interest, while an indirect label could *suggest* the interest. For example, if the user pressed the "Like" button, we have the direct indicator or the interest. However, if the user only clicked at the link, this could be an indicator of *some* interest, but it's an indirect indicator. The user could click by mistake or because the link text was a clickbait, we cannot know for sure. If the label is indirect, this also makes such data less reliable.

Finally, another source of unreliability in the data is **feedback loops**. A feedback loop is a property in the system design when the data used to train the model was obtained using the model itself. Again, imagine that you work on a problem of predicting whether a specific user of a website will like the content, and you only have indirect labels – clicks. If the model is already deployed on the website and the users click on links recommended by the model, this means that the new data (indirectly) reflects not only the interest of users to the content but also how intensively the model recommended that content. If the model decided that a

specific link is very important to recommend to many users, it's likely that more users will click on that link, especially if the recommendation was made repeatedly during several days or weeks.

## 3.5 What Is Good Data

*Good data contains enough information* that can be used for modeling. For example, if you want to build a model that predicts whether the customer will buy a specific product you have to have both the properties of the product in question and the properties of the products customers bought in the past. If you only have the properties of the product and customer's location and name, then the predictions will be likely the same for all users from the same location. If you have enough training examples, then, potentially, the model can derive the gender and ethnicity from the name and make different predictions for men, women, locations, and ethnicities, but not to each customer individually.

*Good data has good coverage* of what you want to do with the model. For example, if you want use the model to classify the web pages by topic and you have a thousand topics that interest you, then your data has to contain examples of document of each of the thousand topics in a quantity sufficient for the algorithm to be able to learn from the data. Imagine a different situation. Let's say that for a certain topic, you only have one or a couple of documents. Let each document contain a unique ID in the text. In such a scenario, the learning algorithm will not be sure what exactly it has to look at in each document to understand to which topic it belongs. Maybe its IDs? They look like good differentiators. If the algorithm decides to use IDs as the main feature to separate these couple examples from the rest of the dataset, then the learned model will not be able to generalize: it will not see these IDs ever again.

*Good data reflects real inputs* that the model will see in production. For example, if you build a system that recognizes cars on the road and all pictures you took were taken during the working hours, then it's unlikely that you will have many examples of nocturne pictures. Once you deploy the model in production, pictures will come from all times during the day and your model will make errors on most nocturne pictures. Also, remember the problem of a cat, a dog, and a raccoon discussed above.

*Good data is as unbiased as possible*. This property can look similar to the previous one, but bias can actually be present in both the data you use for training and the data that the model is applied to in the production environment. A famous example is looking for associations for words using embeddings trained with an algorithm like word2vec. The model predicts that $king - man + women = queen$ but at the same time it predicts that $programmer - man + woman = homemaker$. We will discuss bias in data and how to deal with it in Chapter ??.

Another source of bias can come from human nature. For example, you want to build a model that predicts whether a book will be liked by the readers. You can use the rating users gave to similar books in the past. However, ratings come from unhappy readers more often, because

people usually complain when something is wrong, but rarely do something when everything is fine.

A different example of bias coming from human nature is when you put a poll on the website and ask users rate something. The ratings you receive will be biased because they will represent the opinion of people who tend to participate in online polls and not the general population.

A user interface can also be a source of bias. For example, you want to predict the popularity of a news article and you use the click rate as a feature. If some news article was displayed on the top of the page, the number of clicks it got would often be higher compared to another news article displayed on the bottom, even if the latter is more engaging.

*Good data is not a result of the model itself*. This echoes the problem of the feedback loop discussed above. For example, you cannot build a model that predicts the gender of a person from their name and then use the prediction as a new training example.

Alternatively, if you use the model to decide which email messages are important to the user and you highlight those important messages, then you should not directly take the clicks on those emails as a signal that the email is important. The user might have clicked on them because they were highlighted by the model.

*Good data has consistent labels*. Inconsistency in labeling can come from several sources:

- Different people do labeling according to different criteria. Even if people believe that they use the same criteria, different people often interpret the same criteria differently.
- The definition of some classes evolved over time. This results in a situation when two very similar feature vectors receive two different labels.
- Misinterpretation of user's motives. For example, assume that the user ignored a recommended news article. As a consequence, this news article receives a negative label. However, the motive of the user for ignoring this recommendation might be that they already knew the story and not that they are not interested in the topic of the story.

*Good data is large enough* to allow generalization. Sometimes, nothing can be done to increase the accuracy of the model. No matter how much data you throw on the learning algorithm: the information that is contained in the data has low predictive power for your problem. However, and it happens more often in practice, you can get a very accurate model if you pass from thousands of examples to millions or hundreds of millions. You cannot know how much data you need until you start working on your problem.

## 3.6 Dealing With Interaction Data

**Interaction data** is the data you can collect from the interactions of the user with the system your model support. You are considered lucky if you can gather good data from interactions of the user with the system.

Good interaction data contains information on three aspects:

- *context* of interaction,
- *action* of the user in that context,
- *outcome* of interaction.

As an example assume that you build a search engine and your model reranks search results for each user individually. A reranking model takes as input the list of links returned by the search engine based on keywords provided by the user and outputs another list in which the items of the input list changed order. Usually, a reranked model "knows" something about the user and their preferences and can reorder the generic search results for each user individually according to that user's learned preferences. The context here is the search query and the ten documents presented to the user in a specific order. The action is a click of the user on specific document link. The outcome is how much time the user spent reading the document and whether the user hit "back". Another action is the click on the "next page" link. The intuition is that the ranking was good if the user clicked on some link and spent longtime reading the document. The ranking was not so good if the user clicked a link and then hit "back" quickly. The ranking was bad if the user clicked "next page". This data can be used to improve the ranking algorithm.

## 3.7 Common Problems With Data

The data you work with can have several problems. In this section, I cite the most important of these problems and what you can do to alleviate them.

### 3.7.1 High Cost

Getting unlabeled data can be expensive, however labeling data is the most expensive work, especially if the work is done manually.

Getting unlabeled data is expensive if the data has to be gathered specifically for your problem. Let's say you want to know where different types of commerce are located in a city. The best solution would be to buy this data from a government agency, however, for various reasons it can be complicated or even impossible. Or the data in the government database can be incomplete or outdated. To get the up-to-date data, you may decide to send cars equipped with cameras on the streets of a given city and these cars would take pictures of all building on the streets.

As you can imagine, such an enterprise is not cheap. However, having pictures of all the buildings in the city is not enough. We want to know the type of commerce in every building. Now we need labeled data: pictures of building facades with labels: "coffee house", "bank", "grocery", "drug store", "gas station", etc. These labels have to be assigned manually and

paying someone for doing that work is equally expensive. By the way, Google has had a clever idea to outsource this labeling work to anonymous people as part of its free reCAPTCHA service. reCAPTCHA thus solves two problems: reducing spam on the Web and providing cheap labeled data to Google.

Original photo (Labeled photo)

Figure 3.3: The unlabeled and labeled aerial photo. Credit: Tom Fisk.

In Figure 3.3, you can see how much work has to be done in certain cases to label one image. In this task, the goal is to segment a picture by assigning to every pixel labels from the following list: “heavy truck”, “car or light truck”, “boat”, “building”, and “container”. To label this example I spent about 30 minutes. You can imagine that if there we more objects in the list of labels (for example “motorcycle”, “tree”, “road”, etc), the time would be even longer, and, hence, the cost.


Figure 3.4: An example of simple labeling interface.

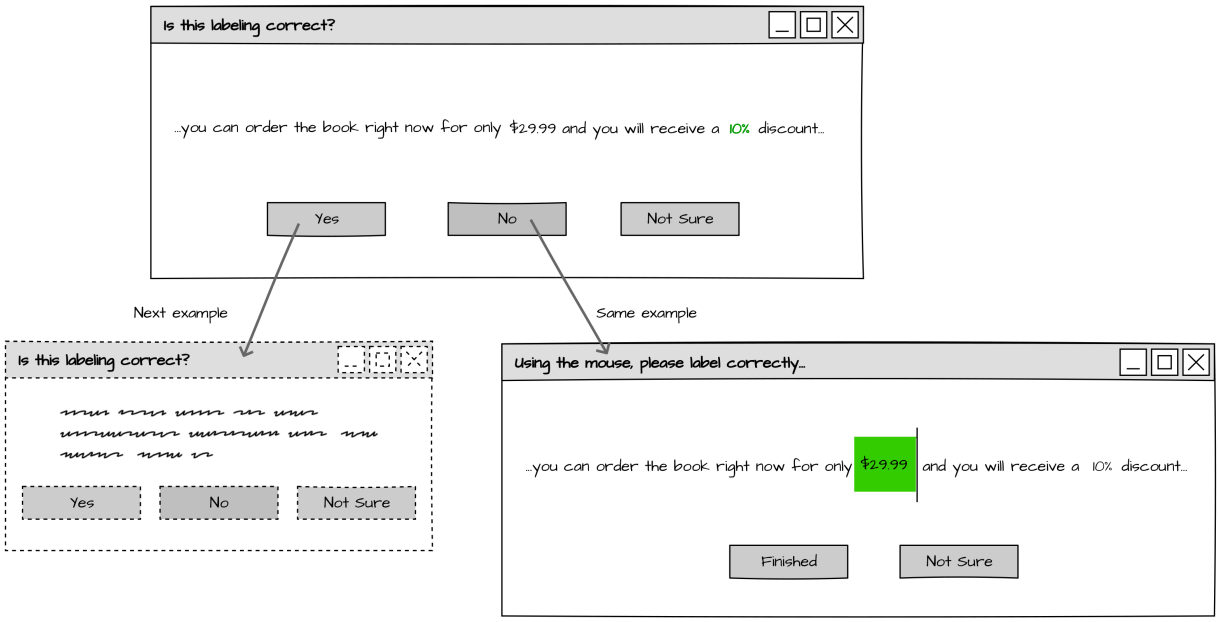


Figure 3.5: An example of noisy pre-labeling workflow.

Well designed labeling tools that minimize the use of the mouse and menus activated by mouse clicks and maximize the use of hotkeys allow reducing the cost and the speed of data labeling.

Whenever possible, reduce the decision the labeler has to make to yes/no decisions. Instead of asking “Find all prices in this text.”, extract all numbers from the text and, then display each number to the labeler one by one in its context and ask “Is this number a price?” as shown in Figure 3.4.

If the labeler clicks the “Not Sure” button, you can save this example to analyze later or simply don’t use such examples for training the model.

Another trick allowing to accelerate labeling is noisy pre-labeling consisting in pre-labeling the example using the current best model. In this scenario, you start by labeling a certain quantity of examples “from scratch”. Then you build the first model that works reasonably well. Then you use the current model to label each new example in place of the human and ask the human labeler whether that automatically assigned label is correct. If the labeler clicks “Yes” then you save this example as usual. If the labeler clicks “No”, then you ask the labeler to label this example manually. See the workflow chart illustrating the process in Figure 3.5.

### 3.7.2 Bad Quality

I already mentioned that the quality of the data is one of the major factors affecting the performance of the model. I cannot stress it strongly enough.

Data quality has two components: raw data quality and labeling quality.

The common problems with raw data:

- Noise
- Bias
- Low predictive power
- Outdated examples
- Outliers
- Leakage

### 3.7.3 Noise

**Noise** in data is a corruption of examples. Images can be blurry or incomplete. Text can lose formatting which makes some words become concatenated or split. Audio data can literally have noise on the background. Poll answers can be incomplete or have missing attributes, such as the responder's age or sex. Noise is usually seen as a random process that corrupts each example independently of other examples in the collection. Sometimes noise is added to feature vectors on purpose, for example in **denoising autoencoders**.

If raw examples have the form of a row in a spreadsheet which has missing attributes, **data imputation** techniques can help in guessing values for those attributes. We will consider data imputation techniques later in this chapter.

Blurred images can be deblurred using specific image deblurring algorithms, though deep machine learning algorithms such as neural networks can learn to deblur if needed. The same can be said about noise in audio data: it can be algorithmically suppressed.

Noise is more a problem when the dataset is relatively small (thousands of example or less) because it can lead to overfitting. In the big data context, noise, if it's randomly applied to each example independently of other examples in the dataset, is typically "averaged out" over multiple examples.

### 3.7.4 Bias

**Bias** in data its inconsistency with the phenomenon it represents. This inconsistency may occur for a number of reasons.

**Selection bias** is the tendency to skew your choice of data sources to those that are easily available, convenient and cost-effective for your purposes. For example, you want to know

the opinion of the readers on your new book. You decide to send several initial chapters of the book to the mailing list subscribers of your previous book and ask them whether they like the new book. It's very likely that the readers of your previous book subscribed to the mailing list will like your new book, however, this information doesn't tell you anything about a typical reader.

**Self-selection bias** is a form of selection bias where you get the data from sources that "volunteered" themselves to provide you that data. Most poll data has this type of bias. For example, you want to train a model that predicts the behavior of successful entrepreneurs. You decide to ask questions to entrepreneurs by letting them decide whether they are successful or not. Then you only keep the data obtained from the entrepreneurs who declared themselves successful. The problem here is that most likely really successful entrepreneurs don't have time to answer your questions, while those who declare themselves successful can be wrong on that matter. Even more

**Omitted variable bias** happens when your featurized data doesn't have a feature important for accurate prediction. For example, let you work on a churn prediction model and you want your model to predict whether a customer of your organization's service will abandon the subscription within a six months period. You build a model and it's accurate enough, however, several weeks after deployment you see many false negatives, much more than expected. You investigate the reason of such a decreased model performance and discover that a new competitor to your organization now offers a very similar service for a lower price. This information wasn't available to your model in the form of a feature, therefore important information for accurate prediction was missing.

**Sponsorship or funding bias** affects the data produced by a sponsored agency. For example, let a famous video game company can sponsor a news agency to produce news about the video game industry. If you try to make a prediction about the video game industry, you might include the news produced by this sponsored agency in your data. It often happens, however, that sponsored news agencies tend to suppress bad news about their sponsor and exaggerate the achievements of their sponsor's products. In this case, your model's performance will be biased because of that sponsorship bias in the data.

**Sampling bias** occurs when the distribution of the examples used to train the model doesn't reflect the distribution of the inputs the model will receive in the production environment. This type of bias often happens in practice. For example, if you are working on a system that classifies documents according to a taxonomy of several hundred topics. You might decide to create a collection of document in which each topic is represented by an equal amount of documents. Once you finish working on the model, you observe the accuracy of 95% which is considered good for your purposes. However, soon after the deployment, you see that the wrong topic is assigned to about 30% of documents (and not 5% as you expected). Why did this happen? One of the possible reasons is the sample bias: it might happen that one or two frequent topics in production data account for about 80% of all input documents. If your model doesn't perform well for these specific frequent topics, then your system will make much more errors in production than you initially expected.

**Prejudice or stereotype bias** is often observed in data obtained from historical sources, such as books or photo archives, or from the online activity of people, such as social media, online forums and comments to online publications. For example, if you use photos from a photo archive to train a model that distinguishes men from women. Because, historically men were frequently taken on a picture in work or outdoor context, while women were more often captured at home and indoor. If you use such biased data, your model will have more difficulty in recognizing a woman outdoors and a man at home.

**Systematic value distortion** is a type of bias that most often occurs when there's a problem with the device making measurements or observations. This type of bias can result in a machine learning model making suboptimal predictions when deploying in the production environment. For example, the training data could be gathered using a camera with a particular setup of white balance which makes white look yellowish. In production, on the other hand, engineers might decide to put a higher quality camera which "sees" white as white. Because your model was trained on yellowish lower quality pictures, the predictions in production for higher-quality input will most likely be suboptimal. This problem is different from the presence of noise in the data. As I mentioned above, noise is considered to be a result of a random process that distorts the data. That means that when you have a sufficiently large dataset, noise can become less a problem because it might average out, as I mentioned above. On the other hand, if the measurements are consistently skewed in one direction all the time, then it can damage data used for training the model, and ultimately generate a poor quality model.

Generally speaking, **confirmation bias** is the tendency to search for, interpret, favor, and recall information in a way that affirms one's prior beliefs or hypotheses. Applied to machine learning, confirmation bias occurs when each example in the dataset is obtained from the answers given by a particular person to a survey, one example per person. Assume that each survey contains multiple questions. The form of those questions can significantly affect the responses. The simplest way for a question to affect the response is to provide response options: "Which kind of pizza do you like: pepperoni, all meats, or vegetarian?". The above way to ask a question doesn't leave the choice of giving an answer different from the three given options. Alternatively, a survey question can be constructed with a particular slant. Instead of asking "Do you recycle?" the analyst with a confirmation bias could ask "Do you dodge from recycling?". In the former case, the person would rather give an honest answer, but less likely in the latter.

Confirmation bias often happens when the analyst was briefed in advance to support a particular conclusion (for example the one in favor of doing "business as usual"). In that situation, analysts can intentionally exclude particular variables from the analysis as unreliable or noisy, for example.

**Labeling bias** happens when labels are assigned to unlabeled examples by a biased process or person. For example, if you ask several people (labelers) to assign a topic to a document by reading the document, some labelers can indeed read the document entirely and assing well-thought labels, while others could just try scan the text, spot some keyphrases and choose the best topic that corresponds the best to the spotted keyphrases. Because each

person's brain pays more attention to key phrases from a specific domain or domains and less to others, the labels assigned by labelers who scan the text without reading will be biased. Alternatively, some labelers would be more interested in reading documents on some topics that they personally prefer. In that case, they might skip uninteresting documents and the latter will be underrepresented in your data.

It is usually impossible to know exactly what biases are present in a dataset. Furthermore, avoiding biases is a challenging task. First of all, be prepared not to be capable of avoiding them entirely. A good habit is to *question everything*: who has created the data you has access to, what were their motivation and quality criteria, and more importantly how and why the data was created. If the data is a result of some research, question the research method and make sure that that research method doesn't contribute to any of the biases described above.

Selection bias is can be avoided by systematically questioning the reason why a specific data source was chosen. If the reason is the simplicity to get the data or the low cost, then you have to pay careful attention to the coverage by that data source of the most important use cases of the model you try to build. If your model has to predict whether a specific customer will subscribe to your new offering, then building the model using *only* the data about your current customers is likely a bad idea, because your existing customers are more loyal to your brand than a random potential customer. Therefore your model will be overly optimistic.

Self-selection bias cannot be completely eliminated. Because it usually happens to survey-like data, the mere cons of the responder to answer the survey's question represents self-selection bias. Usually, the longer the survey, the fewer the chances that the respondent will agree to answer all the questions with a high degree of attention. Therefore, keep your survey short and give an incentive to the responders to give quality answers to the questions. Furthermore, pre-select responders to reduce self-selection. In our example of survey of successful entrepreneurs, don't ask the responder whether they consider themselves successful. Rather, build a list of successful entrepreneurs based on reference from experts or publications in literature, and then only contact people on that list.

It's very hard to completely avoid the omitted variable bias, because, as you know, *we don't know what we don't know*. One possible approach is to use all available information, that is to include into your feature vector as many features as possible, even those you deem unnecessary. This could theoretically make your feature vector very wide (i.e., many dimensions) and sparse (i.e, most dimensions are zero) but if you use a well-tuned regularization your model will decide which features are important and which ones aren't.

Alternatively, if you think that a specific variable is important and leaving it out of your regression model could result in an omitted variable bias, but at the same time you do not have data for it, you can try to use a proxy variable in lieu of the omitted variable. For instance, if you want to build a model that predicts the price of a used car and you cannot have the age of the car, you can use the time of ownership of the car by its current owner. The amount of time the car was owned by the current owner can be taken as a proxy for the age of a car.

Sponsorship bias can be reduced by carefully investigating the source of the data, more

specifically the incentive for the owner of the source to provide the data. For example, it's known that publications on tobacco and pharmaceutical drugs are very often sponsored by either tobacco and pharmaceutical companies or their opponents. The same can be said about news companies, especially those that depend on the advertisement or have an undisclosed business model.

Sampling bias can be avoided by doing research on the real proportion of various properties in the data that will be observed in production and then sampling the data for training by keeping similar proportions.

Prejudice or stereotype bias can be controlled. For our example of distinguishing women from men on pictures, a data analyst could choose to under-sample the number of pictures of women indoor or oversample the number of men at home. In other words, prejudice or stereotype bias is reduced by exposing the learning algorithm to a more even-handed distribution of examples.

Systematic value distortion bias can be alleviated by having multiple measuring devices or hiring humans trained to compare the output of measuring or observing devices.

Confirmation bias can be avoided by letting multiple people validate the questions asked in the survey. The question to ask: "Do I feel uncomfortable or constrained answering this question?". Furthermore, despite more difficulties in analysis, prefer open-ended questions rather than yes/no or multiple-choice questions. If you still prefer to give responders a choice of answers, include the option "Other" and a place to write a different answer.

Labeling bias can be avoided by asking different labelers to label the same example. In case the labels assigned by different labelers to the same example are different, you can ask the labelers why they decided to assign a specific label to an example. If you see that some labelers refer to keyphrases rather than try to paraphrase the whole document, you can identify the labelers who are quickly scanning texts instead of reading them. You can also compare the frequency of skipped document for different labelers. If you see that some labeler skips document frequently than the average, you could ask them why they skipped specific documents: is that because they have a technical problem or they simply are not interested in some topics.

You cannot completely avoid bias in data. There's no silver bullet. As a general rule, keep a human in the loop, especially if your model affects people's lives. There is a temptation among the data analysts to assume that machine learning models are inherently fair because they make decisions based on evidence and math as opposed to messy human judgments. This is, unfortunately, not always the case: inevitably, a model trained on biased data will produce biased results. It is the duty of people building the model to ensure that its outputs are fair. But what's fair, you may ask? Unfortunately, again, there is no silver bullet measurement that would always detect unfairness. Choosing an appropriate definition of model fairness is always problem-specific and requires human judgment.

To keep a human in the loop in all stages of data gathering and preparation is the best approach to make sure that the damage caused by a statistical model is minimized.

### 3.7.5 Low Predictive Power

Low predictive power is that kind of problem which you don't see until you spend too much time fruitlessly trying to build a good model. The difficulty to see that problem is due to the fact that you typically don't know why your model doesn't perform well. Is it because the model is not expressive enough or the data doesn't contain enough information to learn from? You don't know.

Let your problem be to predict whether a user of a music streaming service will like a new song. Your data is the name of the artist, the title of the song and song lyrics, as well as whether the user has this song in their playlist. While such a data may at first seem sufficient, the model you will build based on that data alone will be far from perfect.

First of all, the model will very unlikely score high songs from artists that are not in the user's playlist. Furthermore, many users of streaming services only add to their playlist *some* songs of a specific artist. The preferences of a music lover can be significantly influenced by the song arrangement, choice of instruments, sound effects, tone of voice, and subtle changes in tonality, rhythm, and beat. These are properties of the song that cannot be found in its lyrics, title and the name of the artist. They have to be extracted from the sound file. On the other hand, extracting relevant features from a sound file is challenging. Even with modern neural networks, recommending songs based on how they sound is still considered a hard task for AI. Typically, song recommendations are given by comparing playlists of different users and finding those that are similar in composition.

Another example of data with low predictive power is the photos of the sky made by a telescope. Let's say we want to build a model that would predict where to point the telescope to observe something interesting. Our data are photos of various regions of the sky where something interesting was observed in the past. You can imagine that based on the photos alone it's very unlikely that we will be able to train a model that accurately predicts that something interesting will happen. However, if we add to this data the measurements of various sensors, such as the sensors measuring radiofrequency of the signals coming from different parts of the sky or particle bursts, it is more likely that we will be able to make better predictions.

As I said above, detecting that your data has low predictive power for your problem is very hard. It's especially hard if you work with that a dataset for the first time. If you cannot obtain good no matter how complex your models become, I recommend engineering as many additional features as possible (apply your creativity!), especially by using indirect data sources to enrich feature vectors with additional information.

### 3.7.6 Outdated Examples

Once you build the model and deploy it in production, it usually performs well for some time. The length of this time depends entirely on the phenomenon you are modeling.

Typically, a certain model quality monitoring procedure is deployed in the production environment. Once an erratic behavior is detected, new training data is added to fix the behavior of the model, the model is re-trained and re-deployed.

Often, the cause of the error is explained by a finite set of the training set. In this case, adding additional training examples only solidifies the model. However, in many practical scenarios, the model starts making errors because of the so-called **concept shift**. Concept shift is a fundamental change in the properties of the phenomenon your data represents.

To be more specific, imagine that your model predicts whether a given user will like a given piece of content on a website. You have a history of content consumption for each user as well as some indicator of whether they liked the content or not. This indicator can be direct, such as a "like", or indirect, such as the time the user spent "consuming" the content. Once a user consumes some piece of content you add a new example in your training data and periodically rebuild the model for that user.

The model improves with additional training examples and the predictions for each user improve too. However, at the same time, the preferences of some users change. This can happen because of aging, or because a user discovers something new with time and starts liking it more. After the user's preferences change (we can also say, after the concept shift happens), the examples added to the training data before the concept shift become outdated and actually start hurting the model performance rather than contributing to it.

You can detect a concept shift by regularly building a new test set of recent examples from the production environment and comparing the performance of your model on the recent examples with the historical performance. If you see a decreasing trend in performance on new data, it's an indicator of the concept shift.

In order to remove the outdated examples from the training data, you can first sort your training examples in the order of recency and define an additional hyperparameter &emdash; the percentage of the most recent examples to use for rebuilding the model. You can use **grid search** or another hyperparameter tuning technique to find the best value for that additional hyperparameter by maximizing the performance of the model on a sufficiently representative test set composed of recent examples from the production environment. (We will consider hyperparameter tuning in Chapter ??.)

### 3.7.7 Outliers

Outliers are examples that look dissimilar to a "typical" example from the dataset. It's up to the data analyst to define "dissimilar". Typically, dissimilarity is measured by some distance metric, such as Euclidean distance. In that case, an outlier is an example that lies at an abnormal distance from other examples in the dataset.

In many practical situations, however, what seems to be an outlier in the original feature vector space can be a typical example in a feature vector space transformed using a **kernel**

**function**. Feature space transformation is often explicitly done by a kernel-based algorithm, such as **support vector machine** (SVM), or implicitly by a deep neural network.

Shallow algorithms, such as linear or logistic regression, and some ensemble methods, such as AdaBoost, are particularly sensitive to outliers. SVM has a definition that is less sensitive to outliers: a special penalty hyperparameter regulates the influence of misclassified examples (which often happen to be outliers) on the decision boundary. If this penalty value is low, the SVM algorithm may completely ignore outliers from consideration when drawing the decision boundary. Though, if it's too low, even some normal examples can be ignored. The best value for this hyperparameter has to be found using a hyperparameter tuning technique.

A sufficiently complex neural network can learn to behave differently for each outlier in the dataset, though it's not the desired outcome because the model is unnecessarily complex for the task, which can result in poorer generalization in the production environment.

Whether to exclude outliers from the training data or to use machine learning algorithms robust to outliers is a debated question. Deletion of examples from a dataset is not considered scientifically or methodologically sound, especially in small datasets. In the big data context, on the other hand, outliers don't typically have a significant influence on the model.

From a practical standpoint, if excluding some training examples results in better performance of the model on the holdout data, the exclusion may be justified. Which examples to consider for exclusion can be decided based on a certain dissimilarity measure. A modern approach to getting such a measure is to build an **autoencoder** and use the reconstruction error as the measure of dissimilarity: the higher the reconstruction error for a given example, the more dissimilar it is to the dataset. The hyperparameters of the autoencoder are tuned to minimize the reconstruction error of the holdout data.

### 3.7.8 Leakage

**Data leakage**, also called **target leakage**, is a problem that affects several stages of the machine learning life cycle, from data collection to model evaluation. In this chapter, I will only describe how this problem manifests itself at the data collection and preparation stage. In the subsequent chapters, I will talk about other forms of the problem of data leakage.

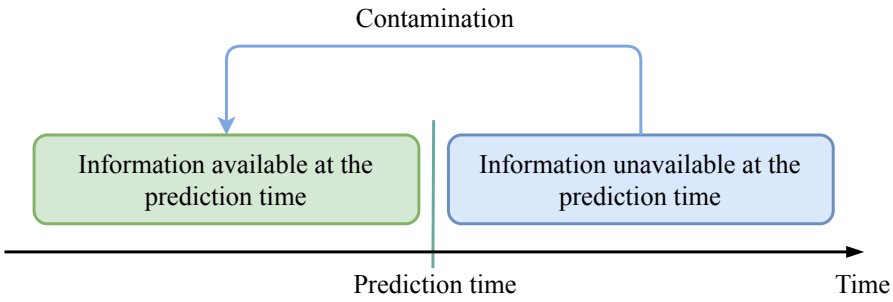


Figure 3.6: Data leakage in a nutshell.

Data leakage in supervised learning is an unintentional introduction of information about the target which should not be available to learn from. It manifests itself in the emergence of unexpected additional information in the training data called the contamination of the training data (Figure 3.6). Training on contaminated data leads to overly optimistic expectations about the model performance in production.

The contamination of the training data can happen at one or several stages of the machine learning life cycle including the stage of data collection and preparation. Below, I consider three typical causes of contamination that can happen at that stage.

## 3.8 Causes of Data Leakage

Let's discuss the three most frequent causes of data leakage at the stage of data collection and preparation.

### 3.8.1 Target is a Function of a Feature

Let our goal be to predict the GDP of the country based on various attributes describing that country: area, population, geographic region, and so on. An example of such data is shown in Figure 3.7. If you don't do a careful analysis of each attribute and how it's related to GDP you might let a leakage happen: in the data in Figure 3.7, two columns, Population, and GDP per capita, multiplied give GDP. The model you will build will most likely perfectly predict GDP by looking at two columns only. The fact that you let GDP be a part of features in slightly modified form (devised by the population) constitutes contamination and is data leakage.

| Country | Population | Region | ... | GDP per capita | GDP |
|---|---|---|---|---|---|
| France | 67M | Europe | ... | 38,800 | 2.6T |
| Germany | 83M | Europe | ... | 44,578 | 3.7T |
| ... | ... | ... | ... | ... | ... |
| China | 1386M | Aisa | ... | 8,802 | 12.2T |

Figure 3.7: An example of the target being a simple function of two features: Population and GDP per capita.

A simpler example of the same kind of data leakage is when you have the target among features in a different format. For example, imagine that you build a model to predict the yearly salary given an employer's attributes. The data you use is a table that contains both monthly and yearly salary among many other attributes of an employee. If you forget to remove the monthly salary from the list of features, this attribute alone will perfectly predict the yearly salary making you believe that your model is perfect until, in production, it doesn't receive information about person's monthly salary, which is likely, otherwise the model would not be needed to its user.

### 3.8.2 Feature Hides the Target

Sometimes the target is not a function of one or more features but rather is "hidden" in one of the features. Consider the dataset shown in Figure 3.8.

| Customer ID | Group | Yearly Spendings | Yearly Pageviews | ... | Gender |
|---|---|---|---|---|---|
| 1 | M18-25 | 1350 | 11,987 | ... | M |
| 2 | F25-35 | 2365 | 8,543 | ... | F |
| ... | ... | ... | ... | ... | ... |
| 18879 | F65+ | 3653 | 6,775 | ... | F |

Figure 3.8: An example of the target being hidden on one of the features.

In this scenario, you use the data about a customer to predict their gender. Look at the column Group. If you closely investigate the data in this column, you will see that Group represents a demographic group to which each existing customer was related. If the data

about customer's gender and age is factual (as opposed to being guessed by another model that would be available in production) then the column Group constitutes a form of data leakage, when the value you want to predict is "hidden" in the value of a feature.

If values in the column Group are predictions provided by another, possibly less accurate, model then you can use this attribute to build a possibly stronger model. This is called **model ensembling** and we will consider this topic in Chapter ??.

### 3.8.3 Feature From the Future

"Feature from the future" is a kind of data leak that is hard to catch if you don't have a clear understanding of the business goal. Imagine a client asked you to build a model that predicts whether a borrower will pay the loan based on the borrower's attributes such as age, gender, education, salary, number, marital status, of children and so on (an example of such data is shown in Figure 3.9).

| Borrower ID | Demographic Group | Education | ... | Late Payment Reminders | Will Pay Loan |
|---|---|---|---|---|---|
| 1 | M35-50 | High school | ... | 0 | Y |
| 2 | F25-35 | Master's | ... | 1 | N |
| ... | ... | ... | ... | ... | ... |
| 65723 | M25-35 | Master's | ... | 3 | N |

Figure 3.9: An example of a feature unavailable at the prediction time: Late Payment Reminders.

You don't make an effort to understand the business context in which your model will be used, you might decide to use all available attributes to predict the value in the column Will Pay Loan, including the data from the column Late Payment Reminders. Your model will most likely work very well and you will send it to the client who will later report that the model doesn't work very well in the production environment.

After close investigation, you will find out that in the production environment, the value of the feature that you created based on the column Late Payment Reminders is always zero. This makes sense because the client uses your model *before* the borrower gets the credit, so no reminders have yet been made! However, your model most likely learned to make the negative prediction once the value of this feature is positive and pays less attention to the values of the other features.

Understanding the business context in which the model will be used is, thus, very important to avoid this kind of data leakage.

# Chapter 4

# Stay Tuned

Stay tuned and subscribe to the mailing list at themlbook.com.

## 4.1 Counterfeit Alert

As of April 2019, the Internet is full of counterfeit copies of my book, including printed copies. To avoid buying counterfeit, I recommend following the links on the book's supporting website themlbook.com. Even if you buy on Amazon, you should make sure that you buy directly from Amazon and not from a third-party seller.

## 4.2 Acknowledgements

I am grateful to these wonderful people for their help: Harpreet Sahota, Vasco Lopes, Kelvin Sundli, Suhel Khan.