Andriy Burkov

MACHINE LEARNING ENGINE--ERING

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 property 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 jump over this subsection.

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 $\{spam, not\_spam\}$.

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.

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. In **dimensionality reduction**, the output of the model is a feature vector that has fewer features than the input $\mathbf{x}$; in **outlier detection**, the output is a real number that indicates how $\mathbf{x}$ is different from a "typical" example in the dataset.

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.

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

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 solving all problem 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 emails from people which 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.

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, building 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 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" a "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

element, making your extraction rule wrong. If the collection of webpages you scrape is large (thousands of webpages), 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/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 more important in combination. 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 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) solution. 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 such 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 and 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 to work 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:

- 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 you will not have to be improved frequently over time,
- you can fill an exhaustive lookup input-output 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 problem definition, to 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 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 engineer 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. 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.

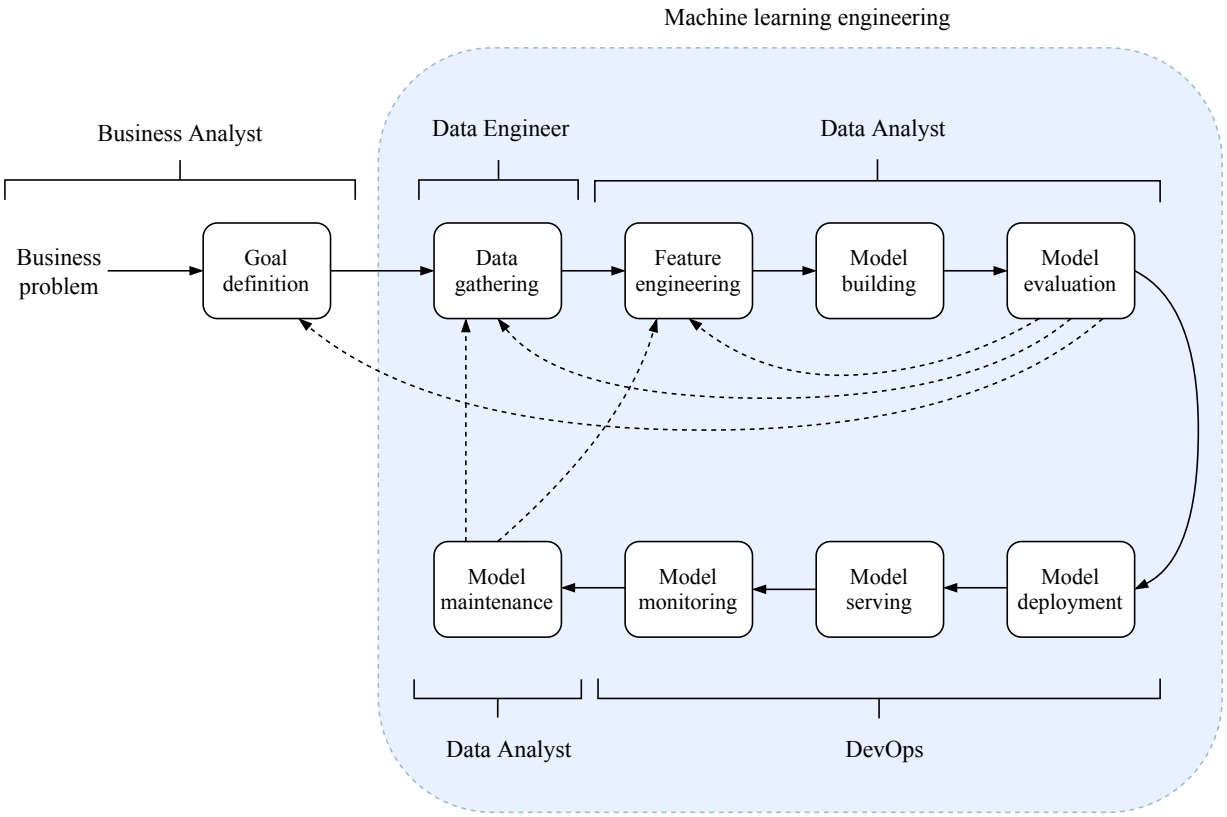

Figure 1.1: Machine learning project life cycle.

1.5 Machine Learning Project Life Cycle

Machine learning project starts with understanding the business objective. Usually, a business analyst works with both the client (or the product owner, if the machine learning project supports a product developed and sold by the company) 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**.

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 the at some stages, a decision to can be made to go back in the process and either gather more data or 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 and how to structure the machine learning team. The next chapter is devoted to these two 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 the following:

- Impact
- 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 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 this 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 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 (assigning labels to unlabeled examples),
- how much examples are needed (usually cannot be known in advance, but can be estimated from 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?

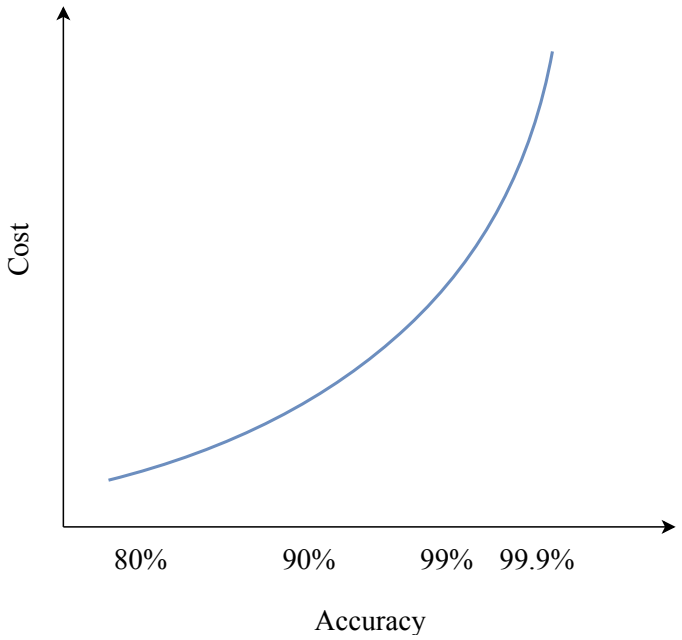

Figure 2.1: Superlinear growth of the cost as a function of accuracy requirement.

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:

- 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% 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 if 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, 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 you can hope that it’s solvable for 1000 classes as well. Your measurements can be used to estimate the time required to solve the complete 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 ne 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 frows 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. It will help not only in reporting but will also simplify the estimation of project complexity in the future.

2.3 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 have to know 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 fast APIs to make sure that the data is easily accessible by analysts and other data consumers. Data engineers are typically 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 Gathering

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