Machine Learning ENGINEERING

Andriy Burkov

"In theory, there is no difference between theory and practice. But in practice, there is."

— Benjamin Brewster

"The perfect project plan is possible if one first documents a list of all the unknowns."

— Bill Langley

"When you're fundraising, it's AI. When you're hiring, it's ML. When you're implementing, it's linear regression. When you're debugging, it's printf()."

— Baron Schwartz

The book is distributed on the "read first, buy later" principle.

1 Introduction

Though the reader of this book should have a basic understanding of machine learning, it is still important to start with definitions, so that we are sure that we have a common understanding of the terms used throughout the book.

Below, I repeat some of the definitions from Chapter 2 of The Hundred-Page Machine Learning Book and also give several new ones. If you read my first book, some parts of this chapter might sound familiar.

After reading this chapter, we will understand the same way such concepts as supervised and unsupervised learning. We will agree on the data terminology, such as data used directly and indirectly, raw and tidy data, training and holdout data.

We will know when to use machine learning, when not to use it, and various forms of machine learning such as model- and instance-based, deep and shallow, classification and regression, and others.

Finally, we will define the scope of machine learning engineering and introduce the machine learning project lifecycle.

1.1 Notation and Definitions

Let's start by stating the basic mathematical notation and define the terms and notions, to which we will often have recourse in this book.

1.1.1 Data Structures

A **scalar**[1] is a simple numerical value, like 15 or -3.25. Variables or constants that take scalar values are denoted by an italic letter, like x or a.

A **vector** is an ordered list of scalar values, called attributes. We denote a vector as a bold character, for example, $\mathbf{x}$ or $\mathbf{w}$. Vectors can be visualized as arrows that point to some directions as well as points in a multi-dimensional space. Illustrations of three two-dimensional vectors, $\mathbf{a} = [2, 3]$, $\mathbf{b} = [-2, 5]$, and $\mathbf{c} = [1, 0]$ are given in Figure 1.1.1. We denote an attribute of a vector as an italic value with an index, like this: $w^{(j)}$ or $x^{(j)}$. The index j denotes a specific **dimension** of the vector, the position of an attribute in the list. For instance, in the vector $\mathbf{a}$ shown in red in Figure 1.1.1, $a^{(1)} = 2$ and $a^{(2)} = 3$.

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

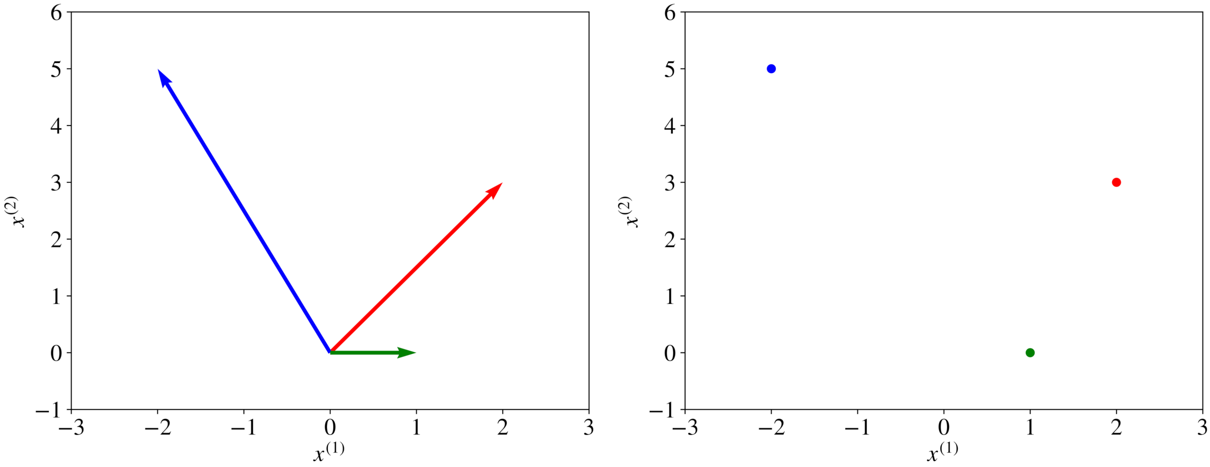

Figure 1: Three vectors visualized as directions and as points.

The notation $x^{(j)}$ should not be confused with the power operator, such as the 2 in x^2 (squared) or 3 in x^3 (cubed). If we want to apply a power operator, say squared, to an indexed attribute of a vector, we write like this: $(x^{(j)})^2$.

A variable can have two or more indices, like this: $x_i^{(j)}$ or like this $x_{i,j}^{(k)}$. For example, in neural networks, we denote as $x_{l,u}^{(j)}$ the input feature j of unit u in layer l.

A **matrix** is a rectangular array of numbers arranged in rows and columns. Below is an example of a matrix with two rows and three columns,

$$\mathbf{A} = \begin{bmatrix} 2 & -2 & 1 \\ 3 & 5 & 0 \end{bmatrix}.$$

Matrices are denoted with bold capital letters, such as $\mathbf{A}$ or $\mathbf{W}$. You can notice from the above example of matrix $\mathbf{A}$ that matrices can be seen as regular structures composed of vectors. Indeed, the columns of matrix $\mathbf{A}$ above are vectors $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ illustrated in Figure .

A **set** is an unordered collection of unique elements. We denote a set as a calligraphic capital character, for example, $\mathcal{S}$. A set of numbers can be finite (include a fixed amount of values). In this case, it is denoted using accolades, for example, $\{1, 3, 18, 23, 235\}$ or $\{x_1, x_2, x_3, x_4, \ldots, x_n\}$. Alternatively, a set can be infinite and include all values in some interval. If a set includes all values between a and b, including a and b, it is denoted using brackets as $[a, b]$. If the set doesn't include the values a and b, such a set is denoted using parentheses like this: (a, b). For example, the set $[0, 1]$ includes such values as 0, 0.0001, 0.25, 0.784, 0.9995, and 1.0. A special set denoted $\mathbb{R}$ includes all numbers from minus infinity to plus infinity.

When an element x belongs to a set $\mathcal{S}$, we write $x \in \mathcal{S}$. We can obtain a new set $\mathcal{S}_3$ as an **intersection** of two sets $\mathcal{S}_1$ and $\mathcal{S}_2$. In this case, we write $\mathcal{S}_3 \leftarrow \mathcal{S}_1 \cap \mathcal{S}_2$. For example $\{1, 3, 5, 8\} \cap \{1, 8, 4\}$ gives the new set $\{1, 8\}$.

We can obtain a new set $\mathcal{S}_3$ as a **union** of two sets $\mathcal{S}_1$ and $\mathcal{S}_2$. In this case, we write $\mathcal{S}_3 \leftarrow \mathcal{S}_1 \cup \mathcal{S}_2$. For example $\{1, 3, 5, 8\} \cup \{1, 8, 4\}$ gives the new set $\{1, 3, 5, 8, 4\}$.

The notation $|\mathcal{S}|$ means the size of set $\mathcal{S}$, that is, the number of elements it contains.

1.1.2 Capital Sigma Notation

The summation over a collection $\mathcal{X} = \{x_1, x_2, \ldots, x_{n-1}, x_n\}$ or over the attributes of a vector $\mathbf{x} = [x^{(1)}, x^{(2)}, \ldots, x^{(m-1)}, x^{(m)}]$ is denoted like this:

$$\sum_{i=1}^{n} x_i \stackrel{\text{def}}{=} x_1 + x_2 + \ldots + x_{n-1} + x_n, \text{ or else: } \sum_{j=1}^{m} x^{(j)} \stackrel{\text{def}}{=} x^{(1)} + x^{(2)} + \ldots + x^{(m-1)} + x^{(m)}.$$

The notation $\stackrel{\text{def}}{=}$ means "is defined as".

The **Euclidean norm** of a vector $\mathbf{x}$, denoted by $\|\mathbf{x}\|$, characterizes the "size" or the "length" of the vector. It's given by $\sqrt{\sum_{j=1}^{D} \left(x^{(j)}\right)^2}$.

The distance between two vectors $\mathbf{a}$ and $\mathbf{b}$ is given by the **Euclidean distance**:

$$\|a - b\| \stackrel{\text{def}}{=} \sqrt{\sum_{i=1}^{N} \left(a^{(i)} - b^{(i)}\right)^2}.$$

1.2 What is Machine Learning

Machine learning is a subfield of computer science that is concerned with building algorithms that, 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) collecting a dataset, and
2) algorithmically training 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. For the same reason, I often say "model" referring to a statistical model.

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

1.2.1 Supervised Learning

In **supervised learning**, the data analyst works with a collection of **labeled examples** $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$. Each element $\mathbf{x}_i$ among N is called a **feature vector**. In computer science, a vector is a one-dimensional array. A one-dimensional array, in turn, is an ordered and indexed sequence of values. The length of that sequence of values, D, is called the vector's **dimensionality**.

A feature vector is a vector in which each dimension j from 1 to D contains a value that describes the example. Each such value is called a **feature** and is denoted as $x^{(j)}$. For instance, if each example $\mathbf{x}$ in our 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 $\mathbf{x}_k$, for all k from 1 to 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 think of 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 and 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**. The value that has to be predicted by a supervised model is called a **target**. An example of regression is a problem of predicting the salary of an employee given their work experience and knowledge. An example of classification is when a doctor enters the characteristics of a patient into a software application, and the application returns the diagnosis.

The difference between classification and regression is shown in Figure 2. In classification, the learning algorithm looks for a line (or, more generally, a hypersurface) that separates examples of different classes from one another. In regression, on the other hand, the learning algorithm looks to find a line or a hypersurface that closely follows the training examples.

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

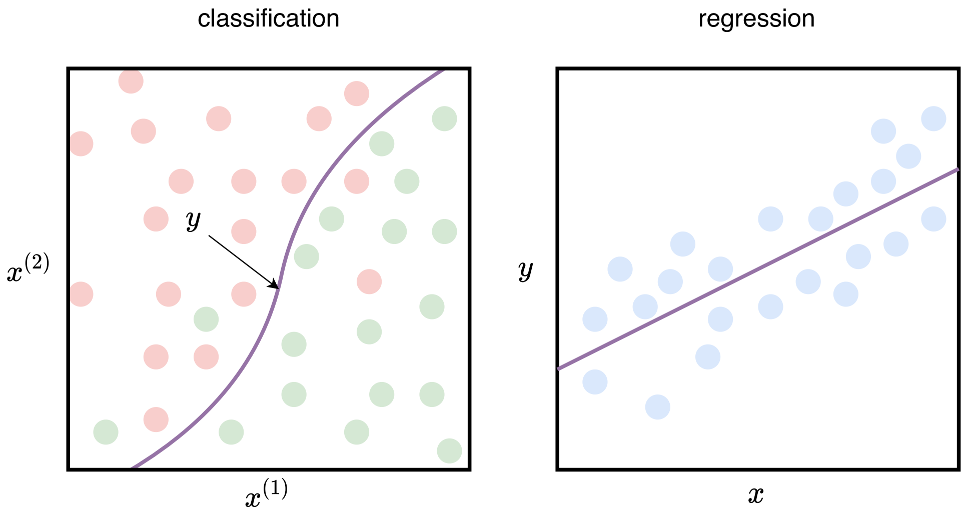

Figure 2: Difference between classification and regression.

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

Even if the model is typically a mathematical function, when thinking about what the 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 experience with similar examples, outputs a value. That output value is a number or a class "the most similar" to the labels seen in the past in the examples with similar values of features. It looks simplistic, but the decision tree model and the k-nearest neighbors algorithm work almost like that.

1.2.2 Unsupervised Learning

In **unsupervised learning**, the dataset is a collection of **unlabeled examples** $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$. 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 a sufficiently representative yet small subset of unlabeled examples from a large collection of examples for manual 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 model's output is a feature vector with fewer dimensions than the input. For example, the scientist has a feature vector that is too complex to visualize (it has more than three dimensions). The dimensionality reduction model can transform that feature vector into a new feature vector (by preserving the information up to some extent) with only two or three dimensions. This new feature vector can be plotted on a graph.

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

1.2.3 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, by using many unlabeled examples, a learning algorithm can find (we might say "produce" or "compute") a better model.

1.2.4 Reinforcement Learning

Reinforcement learning is a subfield of machine learning where the machine (called an agent) "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 non-terminal states. Different actions bring different rewards and could also move the machine to another state of the environment. A common goal of a reinforcement learning algorithm is to learn an optimal **policy**.

An optimal 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 long-term reward.

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

In this book, for simplicity, most explanations are limited to supervised learning. However, all the material presented in the book is applicable to other types of machine learning.

1.3 Data and Machine Learning Terminology

Now let's introduce the common data terminology (such as data used directly and indirectly, raw and tidy data, training and holdout data) and the terminology related to machine learning (such as baseline, hyperparameter, pipeline, and others).

1.3.1 Data Used Directly and Indirectly

The data you will work with in your machine learning project can be used to form the examples $\mathbf{x}$ **directly** or **indirectly**.

Imagine that we build a named entity recognition system. The input of the model is a sequence of words; the output is the sequence of labels[3] 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, which we call a feature vector.[4] 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 in that specific sequence, such as its shape (lowercase, uppercase, capitalized, and so on). Or it can be binary attributes indicating whether this word is the first word of some human name or the last word of the name of some location or organization. To create these latter binary features, we may decide to use some dictionaries, lookup tables, gazetteers, or other machine learning models making predictions about words.

You could already have noticed 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 used indirectly: we can use it to extend feature vectors with additional features, but we cannot use it to create new feature vectors.

1.3.2 Raw and Tidy Data

As we just discussed, directly used data is a collection of entities that constitute the basis of a dataset. Each entity in that collection can be transformed into a training example. **Raw data** is a collection of entities in their natural form; they cannot always be directly employable for machine learning. For instance, a Word document or a JPEG file are pieces of raw data; they cannot be directly used by a machine learning algorithm.[5]

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

[4] The terms "attribute" and "feature" are often used interchangeably. 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.

[5] The term "unstructured data" is often used to designate a data element that contains information whose type was not formally defined. Examples of unstructured data are photos, images, videos, text messages, social media posts, PDFs, text documents, and emails. The term "semi-structured data" refers to data elements whose structure helps deriving types of some information encoded in those data elements. Examples of semi-structured data include log files, comma- and tab-delimited text files, as well as documents in JSON and XML formats.

To be employable in machine learning, a necessary (but not sufficient) condition for the data is to be tidy. **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. Sometimes raw data can be tidy, e.g., provided to you in the form of a spreadsheet. However, in practice, to obtain tidy data from raw data, data analysts often resort to the procedure called **feature engineering**, which is applied to the direct and, optionally, indirect data with the goal to transform each raw example into a feature vector **x**. Chapter 4 is devoted entirely to feature engineering.

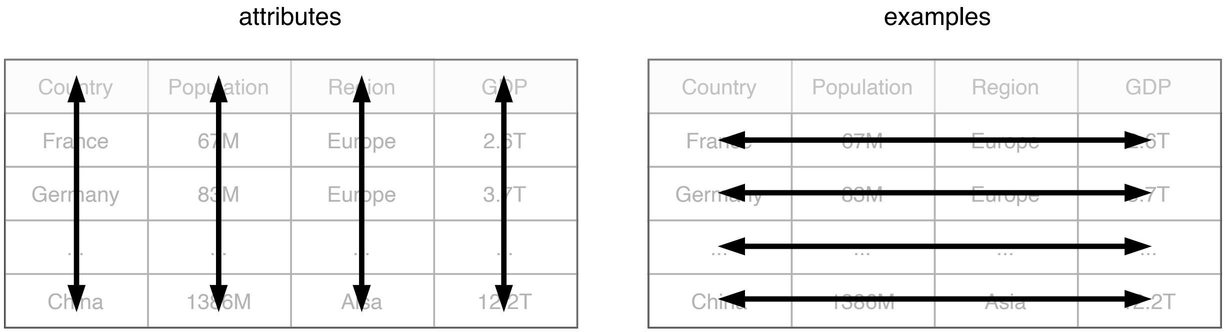

Figure 3: Tidy data: examples are rows and attributes are columns.

It's important to note here that 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**.

The term "tidy data" was coined by Hadley Wickham in his paper with the same title.[6]

As I mentioned at the beginning of this subsection, data can be tidy, but still not usable by a particular machine learning algorithm. Most machine learning algorithms, in fact, only accept training data in the form of a collection of numerical feature vectors. Consider the data shown in Figure 3. 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 Section **??** of Chapter 4, we will see how to transform a categorical attribute into a numerical feature.

Note that in the academic machine learning literature, the word "example" typically refers to a tidy data example with an optionally assigned label. However, during the stage of data collection and labeling, which we consider in the next chapter, examples can still be in the raw form: images, texts, or rows with categorical attributes in a spreadsheet. In this book, when it's important to highlight the difference, I will say **raw example** to indicate that a

[6]Wickham, Hadley. "Tidy data." Journal of Statistical Software 59.10 (2014): 1-23.

piece of data was not transformed into a feature vector yet. Otherwise, assume that examples have the form of feature vectors.

1.3.3 Training and Holdout Sets

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

1) training set,
2) validation set,[7] and
3) test set.

Once you have got the data in the form of a collection of examples, the first thing you do in your machine learning project is shuffle the examples and split the dataset into three distinct sets: **training**, **validation**, and **test**. The training set is usually the biggest one; the learning algorithm uses the training set to produce the model. The validation and test sets are roughly the same size, much smaller than the size of the training set. The learning algorithm is not allowed to use examples from the validation or test sets to train the model. That is why those two sets are also called **holdout sets**.

The reason to have three sets, and not one, is simple: when we train a model, we don't want 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. However, 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. In other words, we want good performance on a holdout set.[8]

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 configuration values for that learning algorithm (known as **hyperparameters**). We use the test set to assess the model before delivering it to the client or putting it in production. That is why it's important to make sure that no information from the validation or test sets is exposed to the learning algorithm. Otherwise, the validation and test results will most likely be too optimistic. This can indeed happen due to **data leakage**, an important phenomenon we consider in Section **??** of Chapter 3 and subsequent chapters.

[7] In some literature, the validation set can also be called "development set." Sometimes, when the labeled examples are scarce, analysts can decide to work without a validation set, as we will see in Chapter 5 in the section on **cross-validation**.

[8] To be precise, we want the model to do well on most random samples from the statistical distribution to which our data belongs. We assume that if the model demonstrates good performance on a holdout set, randomly drawn from the unknown distribution of our data, there are high chances that our model will do well on other random samples of our data.

1.3.4 Baseline

In machine learning, a **baseline** is a simple algorithm for solving a problem, usually based on a heuristic, simple summary statistics, randomization, or very basic machine learning algorithm. For example, if your problem is classification, you can pick a baseline classifier and measure its performance. This baseline performance will then become what you compare any future model to (usually, built using a more sophisticated approach).

1.3.5 Machine Learning Pipeline

A machine learning **pipeline** is a sequence of operations on the dataset that goes from its initial state to the model.

A pipeline can include, among others, such stages as data partitioning, missing data imputation, feature extraction, data augmentation, class imbalance reduction, dimensionality reduction, and model training.

In practice, when we deploy a model in production, we usually deploy an entire pipeline. Furthermore, an entire pipeline is usually optimized when hyperparameters are tuned.

1.3.6 Parameters vs. Hyperparameters

Hyperparameters are inputs of machine learning algorithms or pipelines that influence the performance of the model. They don't belong to the training data and cannot be learned from it. For example, the maximum depth of the tree in the decision tree learning algorithm, the misclassification penalty in support vector machines, k in the k-nearest neighbors algorithm, the target dimensionality in dimensionality reduction, and the choice of the missing data imputation technique are all examples of hyperparameters.

Parameters, on the other hand, are variables that define the model trained by the learning algorithm. Parameters are directly modified by the learning algorithm based on the training data. The goal of learning is to find such values of parameters that make the model optimal in a certain sense. Examples of parameters are w and b in the equation of linear regression $y = wx + b$. In this equation, x is the input of the model, and y is its output (the prediction).

1.3.7 Classification vs. Regression

Classification is a problem of automatically assigning a **label** to an **unlabeled example**. Spam detection is a famous example of classification.

In machine learning, the classification problem is solved by a **classification learning algorithm** that takes a collection of **labeled examples** as inputs and produces a **model** that can take an unlabeled example as input and either directly output a label or output a number that can be used by the analyst to deduce the label. An example of such a number is a probability of an input data element to have a specific label.

In a classification problem, a label is a member of a finite set of **classes**. If the size of the set of classes is two ("sick"/"healthy", "spam"/"not_spam"), we talk about **binary classification** (also called **binomial** in some sources). **Multiclass classification** (also called **multinomial**) is a classification problem with three or more classes.[9]

While some learning algorithms naturally allow for more than two classes, others are by nature binary classification algorithms. There are strategies to turn a binary classification learning algorithm into a multiclass one. I talk about one of them, **one-versus-rest**, in Section **??** of Chapter 6.

Regression is a problem of predicting a real-valued quantity given an unlabeled example. Estimating house price valuation based on house features, such as area, number of bedrooms, location, and so on, is a famous example of regression.

The regression problem is solved by a **regression learning algorithm** that takes a collection of labeled examples as inputs and produces a model that can take an unlabeled example as input and output a target.

1.3.8 Model-Based vs. Instance-Based Learning

Most supervised learning algorithms are **model-based**. A typical **model** is a **support vector machine** (**SVM**). Model-based learning algorithms use the training data to create a model with **parameters** learned from the training data. In SVM, the two parameters are $\mathbf{w}$ (a vector) and b (a real number). After the model is trained, it can be saved on disk while the training data can be discarded.

Instance-based learning algorithms use the whole dataset as the model. One instance-based algorithm frequently used in practice is **k-Nearest Neighbors** (kNN). In classification, to predict a label for an input example, the kNN algorithm looks at the close neighborhood of the input example in the space of feature vectors and outputs the label that it saw most often in this close neighborhood.

1.3.9 Shallow vs. Deep Learning

A **shallow learning** algorithm learns the parameters of the model directly from the features of the training examples. Most machine learning algorithms are shallow. The notorious exceptions are **neural network** learning algorithms, specifically those that build neural networks with more than one **layer** between input and output. Such neural networks are called **deep neural networks**. In deep neural network learning (or, simply, **deep learning**),

[9] There's still one label per example, though.

contrary to shallow learning, most model parameters are learned not directly from the features of the training examples, but from the outputs of the preceding layers.

1.3.10 Training vs. Scoring

When we apply a machine learning algorithm to a dataset in order to obtain a model, we talk about **model training** or simply training.

When we apply a trained model to an input example (or, sometimes, a sequence of examples) in order to obtain a prediction (or, predictions) or to somehow transform an input, we talk about **scoring**.

1.4 When to Use Machine Learning

Machine learning is a powerful tool for solving practical problems. However, like any tool, it should 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.

1.4.1 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 rules to solve it 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 at 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 those conditions 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.

Another difficulty for writing code to solve a problem lies in the fact that humans have a hard time with prediction problems based on input that has too many parameters; it's especially true when those parameters are **correlated** in unknown ways. For example, take the problem of predicting whether a borrower will repay a loan. Hundreds of numbers represent each borrower: age, salary, account balance, frequency of past payments, married or not, number 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 some other numbers.

Writing code that will make such decisions is hard because, even for an expert, it's not clear how to combine, in an optimal way, all the attributes describing a person into a prediction.

1.4.2 When the Problem Is Constantly Changing

Some problems may continuously change with time so that the programming code must be regularly updated. That results in the frustration of software engineers 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 task 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 <div> inside that <p>." If a website owner changes the design of a 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), every day you will have rules that become wrong; you will end up endlessly fixing those rules. Needless to say that very few software engineers would love to do such work on a daily basis.

1.4.3 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 handcrafted "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 cow in a field (I'm simplifying a bit).

Today, perceptive problems are effectively solved using machine learning models, such as neural networks. We consider the problem of training neural networks in Chapter 6.

1.4.4 When It Is an Unstudied Phenomenon

If we need to be able to make predictions of some phenomenon that 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 the patient's genetic and sensory data. 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 patients and predicting which molecule has the highest chance 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 number of examples of historical log records 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 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 readily available examples of expressions of the person's ideas (in the form of online posts, comments, and other activities). Based on those expressions alone, a machine learning model deployed in a social network can recommend the content or other people to connect with.

1.4.5 When the Problem Has a Simple Objective

Machine learning is especially suitable for solving problems that 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 build a model that works as a general video game, like Mario, or a word processing software, like Word. This is due to too many different decisions to make: what to display, where and when, what should happen as a reaction to the user's input, what to write to or read from the hard drive, and so on; getting examples that illustrate all (or even most) of those decisions is practically infeasible.

1.4.6 When It Is Cost-Effective

Three major sources of cost in machine learning are:

- collecting, preparing, and cleaning the data,
- training the model,
- building and running the infrastructure to serve and and monitor the model, as well as labor resources to maintain it.

The cost of training the model includes human labor and, in some cases, the expensive hardware needed to train deep models. Model maintenance includes continuously monitoring the model and collecting additional data to keep the model up to date.

1.5 When Not to Use Machine Learning

There are plenty of problems that cannot be solved using machine learning; it's hard to characterize all of them. Here we only consider several hints.

You probably should not use machine learning when:

- every action of the system or a decision made by it must be explainable,
- every change in the system's behavior compared to its past behavior in a similar situation must be explainable,
- the cost of an error made by the system is too high,
- you want to get to the market as fast as possible,
- getting the right data is too hard or impossible,
- you can solve the problem using traditional software development at a lower cost,
- a simple heuristic would work reasonably well,
- the phenomenon has too many outcomes while you cannot get a sufficient amount of examples to represent them (like in video games or word processing software),
- you build a system that will not have to be improved frequently over time,
- you can manually fill an exhaustive lookup table by providing the expected output for any input (that is, the number of possible input values is not too large, or getting outputs is fast and cheap).

1.6 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 collection, to model training, to making the model available for use by the product or the customers.

Typically, a data analyst[10] is concerned with understanding the business problem, building a model to solve it, and evaluating the model in a restricted development environment. A machine learning engineer, in turn, is concerned with sourcing the data from various systems and locations and preprocessing it, programming features, training an effective model that 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 includes any activity that lets machine learning algorithms be implemented as a part of an effective production system.

In practice, machine learning engineers might be employed in such activities as rewriting a

[10]Since circa 2013, data scientist has become a popular job title. Unfortunately, companies and experts don't have an agreement on the definition of the term. Instead, I use the term "data analyst" by referring to a person capable of applying numerical or statistical analysis to data ready for analysis.

data analyst's code from rather slow R and Python[11] into more efficient Java or C++, scaling this code and making it more robust, packaging the code into an easy-to-deploy versioned package, optimizing the machine learning algorithm to make sure that it generates a model compatible with, and running correctly in, the organization's production environment.

In many organizations, data analysts execute some of the MLE tasks, such as data collection, transformation, and feature engineering. On the other hand, machine learning engineers often execute some of the data analysis tasks, including learning algorithm selection, hyperparameter tuning, and model evaluation.

Working on a machine learning project is different from working on a typical software engineering project. Unlike traditional software, where a program's behavior usually is deterministic, machine learning applications incorporate models whose behavior may naturally degrade over time, or they can start behaving abnormally. Such abnormal behavior of the model might be explained by various reasons, including a fundamental change in the input data or an updated feature extractor that now returns a different distribution of values or values of a different type. They often say that machine learning systems "fail silently." A machine learning engineer must be capable of preventing such failures or, when it's impossible to prevent them entirely, know how to detect and handle them when they happen.

1.7 Machine Learning Project Life Cycle

A machine learning project starts with understanding the business objective. Usually, a business analyst works with the client[12] 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 a broader engineering project, machine learning must first have a well-defined **goal**. The goal of machine learning is a specification of what a 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 in the world. 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 Primary emails from Promotions with accuracy above 90%.

[11] Many scientific modules in Python are indeed implemented in fast C/C++; however, data analyst's own Python code can still be slow.

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

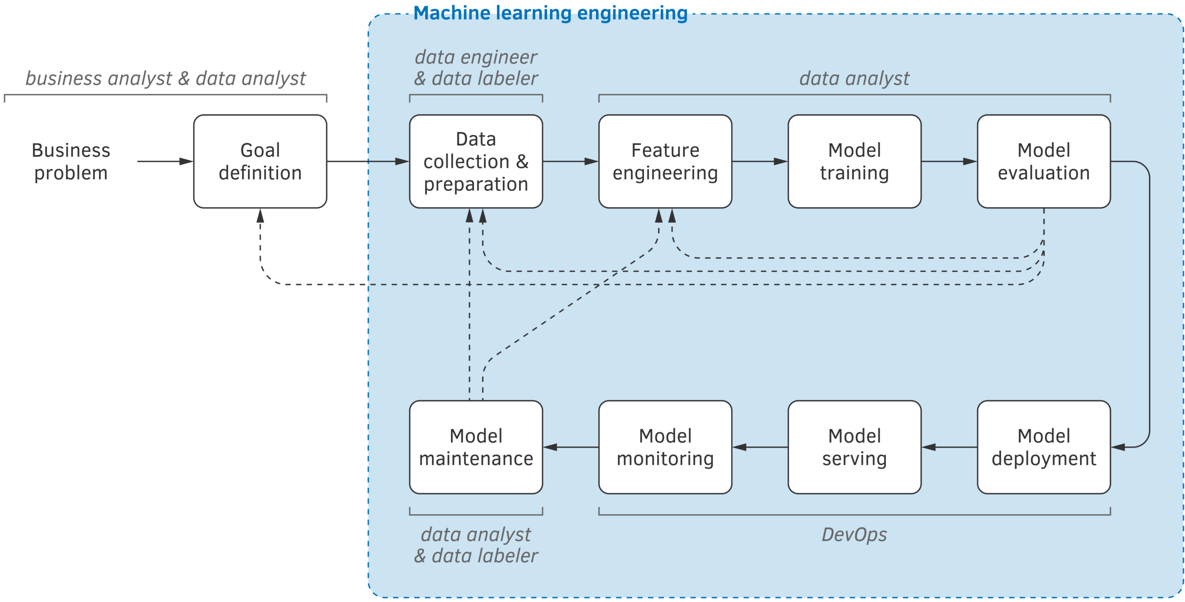

Figure 4: Machine learning project life cycle.

Overall, a machine learning project life cycle, illustrated in Figure 4, consists of the following stages: 1) goal definition, 2) data collection and preparation, 3) feature engineering, 4) model training, 5) model evaluation, 6) model deployment, 7) model serving, 8) model monitoring, and 9) model maintenance.

In Figure 4, the scope of machine learning engineering (and the scope of this book) is limited by the blue zone. The solid arrows show a typical flow of the project stages. The dashed arrows indicate that at some stages, a decision can be made to go back in the process and either collect more data or collect different data, and revise features (by decommissioning some of them and engineering new ones).

Every stage mentioned above will be considered in one of the book's chapters. But first, let's discuss how to prioritize machine learning projects, define the project's goal, and structure a machine learning team. The next chapter is devoted to these three questions.

1.8 Summary

A model-based machine learning algorithm takes a collection of training examples as input and outputs a model. An instance-based machine learning algorithm uses the entire training dataset as a model. The training data is exposed to the machine learning algorithm, while holdout data isn't.

A supervised learning algorithm builds a model that takes a feature vector and outputs a prediction about that feature vector. An unsupervised learning algorithm builds a model that takes a feature vector as input and transforms it into something useful.

Classification is the problem of predicting, for an input example, one of a finite set of classes. Regression, in turn, is a problem of predicting a numerical target.

Data can be used directly or indirectly. Directly-used data is a basis for forming a dataset of examples. Indirectly-used data is used to enrich those examples.

The data for machine learning must be tidy. A tidy dataset can be seen as a spreadsheet where each row is an example, and each column is one of the properties of an example. In addition to being tidy, most machine learning algorithms require numerical data, as opposed to categorical. Feature engineering is the process of transforming data into a form that machine learning algorithms can use.

A baseline is essential to make sure that the model works better than a simple heuristic.

In practice, machine learning is implemented as a pipeline that contains chained stages of data transformation, from data partitioning to missing-data imputation, to class imbalance and dimensionality reduction, to model training. The hyperparameters of the entire pipeline are usually optimized; the entire pipeline can be deployed and used for predictions.

Parameters of the model are optimized by the learning algorithm based on the training data. The values of hyperparameters cannot be learned by the learning algorithm and are, in turn, tuned by using the validation dataset. The test set is only used to assess the model's performance and report it to the client or product owner.

A shallow learning algorithm trains a model that makes predictions directly from the input features. A deep learning algorithm trains a layered model, in which each layer generates outputs by taking the outputs of the preceding layer as inputs.

You should consider using machine learning to solve a business problem when the problem is too complex for coding, the problem is constantly changing, it is a perceptive problem, it is an unstudied phenomenon, the problem has a simple objective, and it is cost-effective.

There are many situations when machine learning should, probably, not be used: when explainability is needed, when errors are intolerable, when traditional software engineering is a less expensive option, when all inputs and outputs can be enumerated and saved in a database, and when data is hard to get or too expensive.

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 collection, to model training, to making the model available for use by the product or the consumers.

A machine learning project life cycle consists of the following stages: 1) goal definition, 2) data collection and preparation, 3) feature engineering, 4) model training, 5) model evaluation, 6) model deployment, 7) model serving, 8) model monitoring, and 9) model maintenance.

Every stage will be considered in one of the book's chapters.