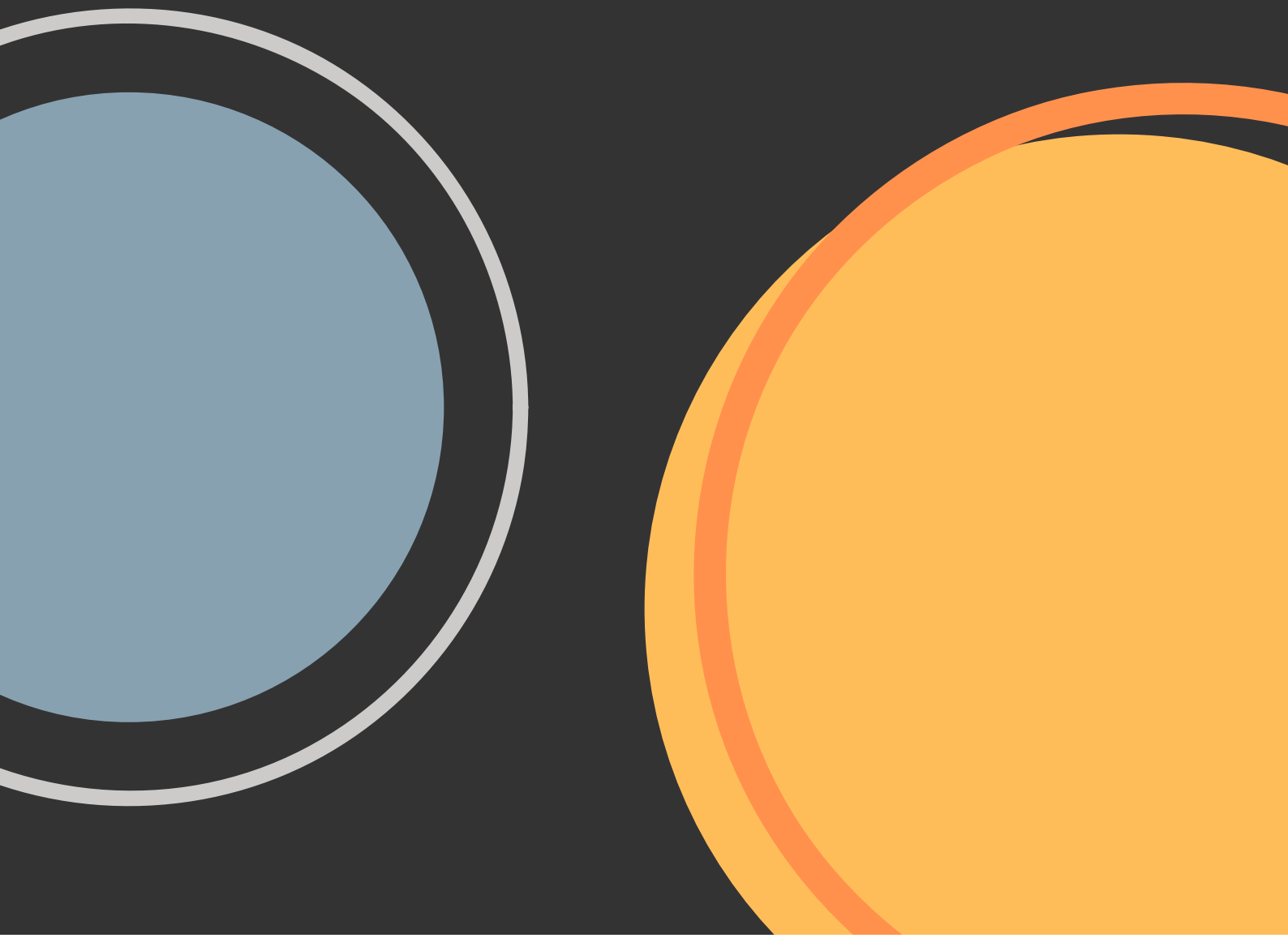
Andriy Burkov's

THE HUNDRED-PAGE MACHINE LEARNING BOOK



Preface

Let's start by telling the truth: machines don't learn. What a typical "learning machine" does, is finding a mathematical formula, which, when applied to a collection of inputs (called "training data"), produces the desired outputs. This mathematical formula also generates the correct outputs for most other inputs (distinct from the training data) on the condition that those inputs come from the same or a similar statistical distribution as the one the training data was drawn from.

Why isn't that learning? Because if you slightly distort the inputs, the output is very likely to become completely wrong. It's not how learning in animals works. If you learned to play a video game by looking straight at the screen, you would still be a good player if someone rotates the screen slightly. A machine learning algorithm, if it was trained by "looking" straight at the screen, unless it was also trained to recognize rotation, will fail to play the game on a rotated screen.

So why the name "machine learning" then? The reason, as is often the case, is marketing: Arthur Samuel, an American pioneer in the field of computer gaming and artificial intelligence, coined the term in 1959 while at IBM. Similarly to how in the 2010s IBM tried to market the term "cognitive computing" to stand out from competition, in the 1960s, IBM used the new cool term "machine learning" to attract both clients and talented employees.

As you can see, just like artificial intelligence is not intelligence, machine learning is not learning. However, machine learning is a universally recognized term that usually refers to the science and engineering of building machines capable of doing various useful things without being explicitly programmed to do so. So, the word "learning" in the term is used by analogy with the learning in animals rather than literally.

Who This Book is For

This book contains only those parts of the vast body of material on machine learning developed since the 1960s that have proven to have a significant practical value. A beginner in machine learning will find in this book just enough details to get a comfortable level of understanding of the field and start asking the right questions.

Practitioners with experience can use this book as a collection of directions for further self-improvement. The book also comes in handy when brainstorming at the beginning of a project, when you try to answer the question whether a given technical or business problem is "machine-learnable" and, if yes, which techniques you should try to solve it.

How to Use This Book

If you are about to start learning machine learning, you should read this book from the beginning to the end. (It's just a hundred pages, not a big deal.) If you are interested in a

1 Introduction

1.1 What is Machine Learning

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.

1.2 Types of Learning

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

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

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

1.2.2 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.

1.2.3 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 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.

1.2.4 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. In this book, I put emphasis on one-shot decision making where input examples are independent of one another and the predictions made in the past. I leave reinforcement learning out of the scope of this book.

1.3 How Supervised Learning Works

In this section, I briefly explain how supervised learning works so that you have the picture of the whole process before we go into detail. I decided to use supervised learning as an example because it's the type of machine learning most frequently used in practice.

The supervised learning process starts with gathering the data. The data for supervised learning is a collection of pairs (input, output). Input could be anything, for example, email messages, pictures, or sensor measurements. Outputs are usually real numbers, or labels (e.g. "spam", "not_spam", "cat", "dog", "mouse", etc). In some cases, outputs are vectors (e.g., four coordinates of the rectangle around a person on the picture), sequences (e.g. ["adjective", "adjective", "noun"] for the input "big beautiful car"), or have some other structure.

Let's say the problem that you want to solve using supervised learning is spam detection. You gather the data, for example, 10,000 email messages, each with a label either "spam" or "not_spam" (you could add those labels manually or pay someone to do that for us). Now, you have to convert each email message into a feature vector.

The data analyst decides, based on their experience, how to convert a real-world entity, such as an email message, into a feature vector. One common way to convert a text into a feature vector, called **bag of words**, is to take a dictionary of English words (let's say it contains 20,000 alphabetically sorted words) and stipulate that in our feature vector:

- the first feature is equal to 1 if the email message contains the word "a"; otherwise, this feature is 0;
- the second feature is equal to 1 if the email message contains the word "aaron"; otherwise, this feature equals 0;
- ...
- the feature at position 20,000 is equal to 1 if the email message contains the word "zulu"; otherwise, this feature is equal to 0.

You repeat the above procedure for every email message in our collection, which gives us 10,000 feature vectors (each vector having the dimensionality of 20,000) and a label ("spam"/"not_spam").

Now you have a machine-readable input data, but the output labels are still in the form of human-readable text. Some learning algorithms require transforming labels into numbers. For example, some algorithms require numbers like 0 (to represent the label "not_spam") and 1 (to represent the label "spam"). The algorithm I use to illustrate supervised learning is called **Support Vector Machine** (SVM). This algorithm requires that the positive label (in our case it's "spam") has the numeric value of +1 (one), and the negative label ("not_spam") has the value of −1 (minus one).

At this point, you have a **dataset** and a **learning algorithm**, so you are ready to apply the learning algorithm to the dataset to get the **model**.

SVM sees every feature vector as a point in a high-dimensional space (in our case, space

is 20,000-dimensional). The algorithm puts all feature vectors on an imaginary 20,000-dimensional plot and draws an imaginary 19,999-dimensional line (a *hyperplane*) that separates examples with positive labels from examples with negative labels. In machine learning, the boundary separating the examples of different classes is called the **decision boundary**.

The equation of the hyperplane is given by two **parameters**, a real-valued vector $\mathbf{w}$ of the same dimensionality as our input feature vector $\mathbf{x}$, and a real number b like this:

$$\mathbf{wx} - b = 0,$$

where the expression $\mathbf{wx}$ means $w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \ldots + w^{(D)}x^{(D)}$, and D is the number of dimensions of the feature vector $\mathbf{x}$.

(If some equations aren't clear to you right now, in Chapter 2 we revisit the math and statistical concepts necessary to understand them. For the moment, try to get an intuition of what's happening here. It all becomes more clear after you read the next chapter.)

Now, the predicted label for some input feature vector $\mathbf{x}$ is given like this:

$$y = \text{sign}(\mathbf{wx} - b),$$

where sign is a mathematical operator that takes any value as input and returns $+1$ if the input is a positive number or -1 if the input is a negative number.

The goal of the learning algorithm — SVM in this case — is to leverage the dataset and find the optimal values $\mathbf{w}^*$ and b^* for parameters $\mathbf{w}$ and b. Once the learning algorithm identifies these optimal values, the **model** $f(\mathbf{x})$ is then defined as:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^*\mathbf{x} - b^*)$$

Therefore, to predict whether an email message is spam or not spam using an SVM model, you have to take a text of the message, convert it into a feature vector, then multiply this vector by $\mathbf{w}^*$, subtract b^* and take the sign of the result. This will give us the prediction ($+1$ means "spam", -1 means "not_spam").

Now, how does the machine find $\mathbf{w}^*$ and b^*? It solves an optimization problem. Machines are good at optimizing functions under constraints.

So what are the constraints we want to satisfy here? First of all, we want the model to predict the labels of our 10,000 examples correctly. Remember that each example $i = 1, \ldots, 10000$ is given by a pair $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i$ is the feature vector of example i and y_i is its label that takes values either -1 or $+1$. So the constraints are naturally:

$$\begin{aligned} \mathbf{wx}_i - b \geq +1 \quad &\text{if } y_i = +1, \\ \mathbf{wx}_i - b \leq -1 \quad &\text{if } y_i = -1. \end{aligned}$$

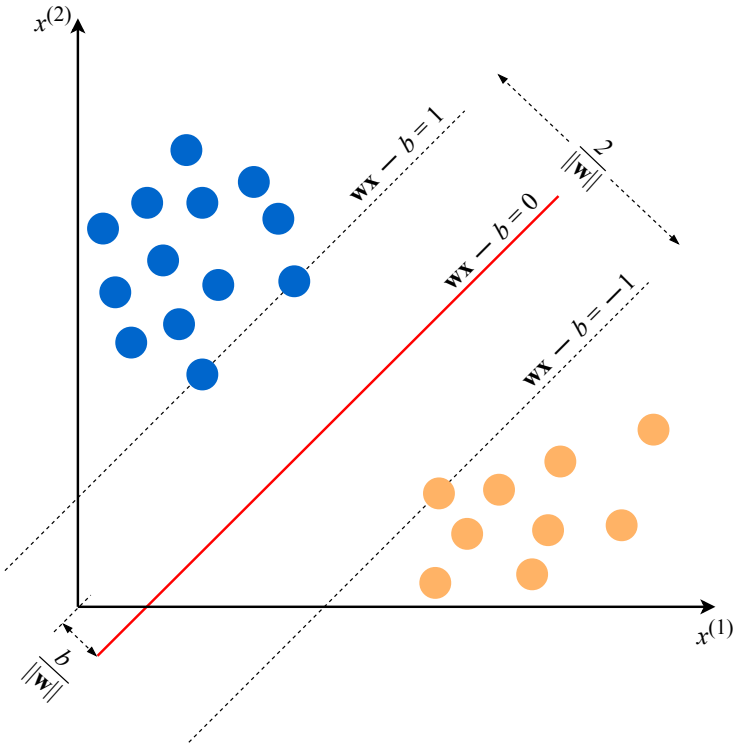


Figure 1: An example of an SVM model for two-dimensional feature vectors.

We would also prefer that the hyperplane separates positive examples from negative ones with the largest **margin**. The margin is the distance between the closest examples of two classes, as defined by the decision boundary. A large margin contributes to a better **generalization**, that is how well the model will classify new examples in the future. To achieve that, we need to minimize the Euclidean norm of $\mathbf{w}$ denoted by $\|\mathbf{w}\|$ and given by $\sqrt{\sum_{j=1}^{D}(w^{(j)})^2}$.

So, the optimization problem that we want the machine to solve looks like this:

Minimize $\|\mathbf{w}\|$ *subject to* $y_i(\mathbf{wx}_i - b) \geq 1$ *for* $i = 1, \ldots, N$. The expression $y_i(\mathbf{wx}_i - b) \geq 1$ is just a compact way to write the above two constraints.

The solution of this optimization problem, given by $\mathbf{w}^*$ and b^*, is called the **statistical model**, or, simply, the **model**. The process of building the model is called **training**.

For two-dimensional feature vectors, the problem and the solution can be visualized as shown in Figure 1. The blue and orange circles represent, respectively, positive and negative examples, and the line given by $\mathbf{wx} - b = 0$ is the decision boundary.

Why, by minimizing the norm of $\mathbf{w}$, do we find the highest margin between the two classes? Geometrically, the equations $\mathbf{wx} - b = 1$ and $\mathbf{wx} - b = -1$ define two parallel hyperplanes, as you see in Figure 1. The distance between these hyperplanes is given by $\frac{2}{\|\mathbf{w}\|}$, so the smaller

the norm $\|\mathbf{w}\|$, the larger the distance between these two hyperplanes.

That's how Support Vector Machines work. This particular version of the algorithm builds the so-called *linear model*. It's called linear because the decision boundary is a straight line (or a plane, or a hyperplane). SVM can also incorporate **kernels** that can make the decision boundary arbitrarily non-linear. In some cases, it could be impossible to perfectly separate the two groups of points because of noise in the data, errors of labeling, or **outliers** (examples very different from a "typical" example in the dataset). Another version of SVM can also incorporate a penalty hyperparameter[3] for misclassification of training examples of specific classes. We study the SVM algorithm in more detail in Chapter 3.

At this point, you should retain the following: any classification learning algorithm that builds a model implicitly or explicitly creates a decision boundary. The decision boundary can be straight, or curved, or it can have a complex form, or it can be a superposition of some geometrical figures. The form of the decision boundary determines the **accuracy** of the model (that is the ratio of examples whose labels are predicted correctly). The form of the decision boundary, the way it is algorithmically or mathematically computed based on the training data, differentiates one learning algorithm from another.

In practice, there are two other essential differentiators of learning algorithms to consider: speed of model building and prediction processing time. In many practical cases, you would prefer a learning algorithm that builds a less accurate model fast. Additionally, you might prefer a less accurate model that is much quicker at making predictions.

1.4 Why the Model Works on New Data

Why is a machine-learned model capable of predicting correctly the labels of new, previously unseen examples? To understand that, look at the plot in Figure 1. If two classes are separable from one another by a decision boundary, then, obviously, examples that belong to each class are located in two different subspaces which the decision boundary creates.

If the examples used for training were selected randomly, independently of one another, and following the same procedure, then, statistically, it is *more likely* that the new negative example will be located on the plot somewhere not too far from other negative examples. The same concerns the new positive example: it will *likely* come from the surroundings of other positive examples. In such a case, our decision boundary will still, *with high probability*, separate well new positive and negative examples from one another. For other, *less likely situations*, our model will make errors, but because such situations are less likely, the number of errors will likely be smaller than the number of correct predictions.

Intuitively, the larger is the set of training examples, the more unlikely that the new examples will be dissimilar to (and lie on the plot far from) the examples used for training.

[3] A hyperparameter is a property of a learning algorithm, usually (but not always) having a numerical value. That value influences the way the algorithm works. Those values aren't learned by the algorithm itself from data. They have to be set by the data analyst before running the algorithm.

To minimize the probability of making errors on new examples, the SVM algorithm, by looking for the largest margin, explicitly tries to draw the decision boundary in such a way that it lies as far as possible from examples of both classes.

The reader interested in knowing more about the *learnability* and understanding the close relationship between the model error, the size of the training set, the form of the mathematical equation that defines the model, and the time it takes to build the model is encouraged to read about the *PAC learning*. The PAC (for "probably approximately correct") learning theory helps to analyze whether and under what conditions a learning algorithm will probably output an approximately correct classifier.