1 A Tutorial Introduction

This chapter describes the central ideas of Support Vector (SV) learning in a nutshell. Its goal is to provide an overview of the basic concepts.

Overview

One such concept is that of a kernel. Rather than going immediately into mathematical detail, we introduce kernels informally as similarity measures that arise from a particular representation of patterns (Section 1.1), and describe a simple kernel algorithm for pattern recognition (Section 1.2). Following this, we report some basic insights from statistical learning theory, the mathematical theory that underlies SV learning (Section 1.3). Finally, we briefly review some of the main kernel algorithms, namely Support Vector Machines (SVMs) (Sections 1.4 to 1.6) and kernel principal component analysis (Section 1.7).

Prerequisites

We have aimed to keep this introductory chapter as basic as possible, whilst giving a fairly comprehensive overview of the main ideas that will be discussed in the present book. After reading it, readers should be able to place all the remaining material in the book in context and judge which of the following chapters is of particular interest to them.

As a consequence of this aim, most of the claims in the chapter are not proven. Abundant references to later chapters will enable the interested reader to fill in the gaps at a later stage, without losing sight of the main ideas described presently.

1.1 Data Representation and Similarity

One of the fundamental problems of learning theory is the following: suppose we are given two classes of objects. We are then faced with a new object, and we have to assign it to one of the two classes. This problem can be formalized as follows:

Training Data

we are given empirical data

$$(x_1, y_1), \ldots, (x_m, y_m) \in \mathcal{X} \times \{\pm 1\}. \tag{1.1}$$

Here, $\mathcal{X}$ is some nonempty set from which the *patterns* x_i (sometimes called *cases*, *inputs*, *instances*, or *observations*) are taken, usually referred to as the *domain*; the y_i are called *labels*, *targets*, *outputs* or sometimes also *observations*.[1] Note that there are

1. Note that we use the term pattern to refer to individual observations. A (smaller) part of the existing literature reserves the term for a generic *prototype* which underlies the data. The

only two classes of patterns. For the sake of mathematical convenience, they are labelled by $+1$ and -1, respectively. This is a particularly simple situation, referred to as *(binary) pattern recognition* or *(binary) classification*.

It should be emphasized that the patterns could be just about anything, and we have made no assumptions on $\mathcal{X}$ other than it being a set. For instance, the task might be to categorize sheep into two classes, in which case the patterns x_i would simply be sheep.

In order to study the problem of learning, however, we need an additional type of structure. In learning, we want to be able to *generalize* to unseen data points. In the case of pattern recognition, this means that given some new pattern $x \in \mathcal{X}$, we want to predict the corresponding $y \in \{\pm 1\}$.[2] By this we mean, loosely speaking, that we choose y such that (x, y) is in some sense similar to the training examples (1.1). To this end, we need notions of *similarity* in $\mathcal{X}$ and in $\{\pm 1\}$.

Characterizing the similarity of the outputs $\{\pm 1\}$ is easy: in binary classification, only two situations can occur: two labels can either be identical or different. The choice of the similarity measure for the inputs, on the other hand, is a deep question that lies at the core of the field of machine learning.

Let us consider a similarity measure of the form

$$\begin{aligned} k &: \mathcal{X} \times \mathcal{X} \to \mathbb{R} \\ &(x, x') \mapsto k(x, x'), \end{aligned} \tag{1.2}$$

that is, a function that, given two patterns x and x', returns a real number characterizing their similarity. Unless stated otherwise, we will assume that k is *symmetric*, that is, $k(x, x') = k(x', x)$ for all $x, x' \in \mathcal{X}$. For reasons that will become clear later (cf. Remark 2.16), the function k is called a *kernel* [359, 4, 42, 62, 223].

General similarity measures of this form are rather difficult to study. Let us therefore start from a particularly simple case, and generalize it subsequently. A simple type of similarity measure that is of particular mathematical appeal is a

Dot Product

dot product. For instance, given two vectors $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^N$, the *canonical dot product* is defined as

$$\langle \mathbf{x}, \mathbf{x}' \rangle := \sum_{i=1}^{N} [\mathbf{x}]_i [\mathbf{x}']_i. \tag{1.3}$$

Here, $[\mathbf{x}]_i$ denotes the ith entry of $\mathbf{x}$.

Note that the dot product is also referred to as *inner product* or *scalar product*, and sometimes denoted with round brackets and a dot, as $(\mathbf{x} \cdot \mathbf{x}')$ — this is where the "dot" in the name comes from. In Section B.2, we give a general definition of dot products. Usually, however, it is sufficient to think of dot products as (1.3).

---

latter is probably closer to the original meaning of the term, however we decided to stick with the present usage, which is more common in the field of machine learning.

2. Doing this for every $x \in \mathcal{X}$ amounts to estimating a *function* $f : \mathcal{X} \to \{\pm 1\}$.

Length

The geometric interpretation of the canonical dot product is that it computes the cosine of the angle between the vectors $\mathbf{x}$ and $\mathbf{x}'$, provided they are normalized to length 1. Moreover, it allows computation of the *length* (or *norm*) of a vector $\mathbf{x}$ as

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}. \tag{1.4}$$

Likewise, the distance between two vectors is computed as the length of the difference vector. Therefore, being able to compute dot products amounts to being able to carry out all geometric constructions that can be formulated in terms of angles, lengths and distances.

Note, however, that the dot product approach is not really sufficiently general to deal with many interesting problems.

- First, we have deliberately not made the assumption that the patterns actually exist in a dot product space. So far, they could be any kind of object. In order to be able to use a dot product as a similarity measure, we therefore first need to represent the patterns as vectors in some dot product space $\mathcal{H}$ (which need not coincide with $\mathbb{R}^N$). To this end, we use a map

$$\begin{aligned} \Phi : \mathcal{X} &\to \mathcal{H} \\ x &\mapsto \mathbf{x} := \Phi(x). \end{aligned} \tag{1.5}$$

- Second, even if the original patterns exist in a dot product space, we may still want to consider more general similarity measures obtained by applying a map (1.5). In that case, Φ will typically be a nonlinear map. An example that we will consider in Chapter 2 is a map which computes products of entries of the input patterns.

Feature Space

In both the above cases, the space $\mathcal{H}$ is called a *feature space*. Note that we have used a bold face $\mathbf{x}$ to denote the vectorial representation of x in the feature space. We will follow this convention throughout the book.

To summarize, embedding the data into $\mathcal{H}$ via Φ has three benefits:

1. It lets us define a similarity measure from the dot product in $\mathcal{H}$,

$$k(x, x') := \langle \mathbf{x}, \mathbf{x}' \rangle = \langle \Phi(x), \Phi(x') \rangle . \tag{1.6}$$

2. It allows us to deal with the patterns geometrically, and thus lets us study learning algorithms using linear algebra and analytic geometry.

3. The freedom to choose the mapping Φ will enable us to design a large variety of similarity measures and learning algorithms. This also applies to the situation where the inputs x_i already exist in a dot product space. In that case, we *might* directly use the dot product as a similarity measure. However, nothing prevents us from first applying a possibly nonlinear map Φ to change the representation into one that is more suitable for a given problem. This will be elaborated in Chapter 2, where the theory of kernels is developed in more detail.

1.2 A Simple Pattern Recognition Algorithm

We are now in the position to describe a pattern recognition learning algorithm that is arguably one of the simplest possible. We make use of the structure introduced in the previous section; that is, we assume that our data are embedded into a dot product space $\mathcal{H}$.[3] Using the dot product, we can measure distances in this space. The basic idea of the algorithm is to assign a previously unseen pattern to the class with closer mean.

We thus begin by computing the means of the two classes in feature space;

$$\mathbf{c}_+ = \frac{1}{m_+} \sum_{\{i|y_i=+1\}} \mathbf{x}_i, \tag{1.7}$$

$$\mathbf{c}_- = \frac{1}{m_-} \sum_{\{i|y_i=-1\}} \mathbf{x}_i, \tag{1.8}$$

where m_+ and m_- are the number of examples with positive and negative labels, respectively. We assume that both classes are non-empty, thus $m_+, m_- > 0$. We assign a new point $\mathbf{x}$ to the class whose mean is closest (Figure 1.1). This geometric construction can be formulated in terms of the dot product $\langle \cdot, \cdot \rangle$. Half way between $\mathbf{c}_+$ and $\mathbf{c}_-$ lies the point $\mathbf{c} := (\mathbf{c}_+ + \mathbf{c}_-)/2$. We compute the class of $\mathbf{x}$ by checking whether the vector $\mathbf{x} - \mathbf{c}$ connecting $\mathbf{c}$ to $\mathbf{x}$ encloses an angle smaller than $\pi/2$ with the vector $\mathbf{w} := \mathbf{c}_+ - \mathbf{c}_-$ connecting the class means. This leads to

$$\begin{aligned} y &= \operatorname{sgn} \langle (\mathbf{x} - \mathbf{c}), \mathbf{w} \rangle \\ &= \operatorname{sgn} \langle (\mathbf{x} - (\mathbf{c}_+ + \mathbf{c}_-)/2), (\mathbf{c}_+ - \mathbf{c}_-) \rangle \\ &= \operatorname{sgn} (\langle \mathbf{x}, \mathbf{c}_+ \rangle - \langle \mathbf{x}, \mathbf{c}_- \rangle + b). \end{aligned} \tag{1.9}$$

Here, we have defined the offset

$$b := \frac{1}{2}(\|\mathbf{c}_-\|^2 - \|\mathbf{c}_+\|^2), \tag{1.10}$$

with the norm $\|\mathbf{x}\| := \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$. If the class means have the same distance to the origin, then b will vanish.

Note that (1.9) induces a decision boundary which has the form of a hyperplane (Figure 1.1); that is, a set of points that satisfy a constraint expressible as a linear equation.

It is instructive to rewrite (1.9) in terms of the input patterns x_i, using the kernel k to compute the dot products. Note, however, that (1.6) only tells us how to compute the dot products between vectorial representations $\mathbf{x}_i$ of inputs x_i. We therefore need to express the vectors $\mathbf{c}_i$ and $\mathbf{w}$ in terms of $\mathbf{x}_1, \ldots, \mathbf{x}_m$.

Decision Function

To this end, substitute (1.7) and (1.8) into (1.9) to get the *decision function*

3. For the definition of a dot product space, see Section B.2.