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.

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.

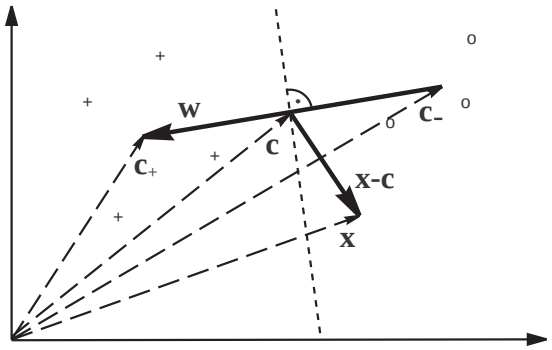


Figure 1.1 A simple geometric classification algorithm: given two classes of points (depicted by 'o' and '+'), compute their means $\mathbf{c}_+, \mathbf{c}_-$ and assign a test pattern $\mathbf{x}$ to the one whose mean is closer. This can be done by looking at the dot product between $\mathbf{x} - \mathbf{c}$ (where $\mathbf{c} = (\mathbf{c}_+ + \mathbf{c}_-)/2$) and $\mathbf{w} := \mathbf{c}_+ - \mathbf{c}_-$, which changes sign as the enclosed angle passes through $\pi/2$. Note that the corresponding decision boundary is a hyperplane (the dotted line) orthogonal to $\mathbf{w}$.

$$\begin{aligned} y &= \operatorname{sgn}\left(\frac{1}{m_+}\sum_{\{i|y_i=+1\}}\langle \mathbf{x}, \mathbf{x}_i\rangle - \frac{1}{m_-}\sum_{\{i|y_i=-1\}}\langle \mathbf{x}, \mathbf{x}_i\rangle + b\right) \\ &= \operatorname{sgn}\left(\frac{1}{m_+}\sum_{\{i|y_i=+1\}}k(x, x_i) - \frac{1}{m_-}\sum_{\{i|y_i=-1\}}k(x, x_i) + b\right). \end{aligned} \tag{1.11}$$

Similarly, the offset becomes

$$b := \frac{1}{2}\left(\frac{1}{m_-^2}\sum_{\{(i,j)|y_i=y_j=-1\}}k(x_i, x_j) - \frac{1}{m_+^2}\sum_{\{(i,j)|y_i=y_j=+1\}}k(x_i, x_j)\right). \tag{1.12}$$

Surprisingly, it turns out that this rather simple-minded approach contains a well-known statistical classification method as a special case. Assume that the class means have the same distance to the origin (hence $b = 0$, cf. (1.10)), and that k can be viewed as a probability density when one of its arguments is fixed. By this we mean that it is positive and has unit integral,[4]

$$\int_{\mathcal{X}} k(x, x')dx = 1 \text{ for all } x' \in \mathcal{X}. \tag{1.13}$$

In this case, (1.11) takes the form of the so-called Bayes classifier separating the two classes, subject to the assumption that the two classes of patterns were generated by sampling from two probability distributions that are correctly estimated by the

4. In order to state this assumption, we have to require that we can define an integral on $\mathcal{X}$.

Parzen windows estimators of the two class densities,

$$p_+(x) := \frac{1}{m_+} \sum_{\{i|y_i=+1\}} k(x, x_i) \text{ and } p_-(x) := \frac{1}{m_-} \sum_{\{i|y_i=-1\}} k(x, x_i), \tag{1.14}$$

Parzen Windows

where $x \in \mathcal{X}$.

Given some point x, the label is then simply computed by checking which of the two values $p_+(x)$ or $p_-(x)$ is larger, which leads directly to (1.11). Note that this decision is the best we can do if we have no prior information about the probabilities of the two classes.

The classifier (1.11) is quite close to the type of classifier that this book deals with in detail. Both take the form of kernel expansions on the input domain,

$$y = \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_i k(x, x_i) + b\right). \tag{1.15}$$

In both cases, the expansions correspond to a separating hyperplane in a feature space. In this sense, the α_i can be considered a *dual representation* of the hyperplane's normal vector [223]. Both classifiers are example-based in the sense that the kernels are centered on the training patterns; that is, one of the two arguments of the kernel is always a training pattern. A test point is classified by comparing it to all the training points that appear in (1.15) with a nonzero weight.

More sophisticated classification techniques, to be discussed in the remainder of the book, deviate from (1.11) mainly in the selection of the patterns on which the kernels are centered and in the choice of weights α_i that are placed on the individual kernels in the decision function. It will no longer be the case that *all* training patterns appear in the kernel expansion, and the weights of the kernels in the expansion will no longer be uniform within the classes — recall that in the current example, cf. (1.11), the weights are either $(1/m_+)$ or $(-1/m_-)$, depending on the class to which the pattern belongs.

In the feature space representation, this statement corresponds to saying that we will study normal vectors $\mathbf{w}$ of decision hyperplanes that can be represented as general linear combinations (i.e., with non-uniform coefficients) of the training patterns. For instance, we might want to remove the influence of patterns that are very far away from the decision boundary, either since we expect that they will not improve the generalization error of the decision function, or since we would like to reduce the computational cost of evaluating the decision function (cf. (1.11)). The hyperplane will then only depend on a subset of training patterns called *Support Vectors*.

1.3 Some Insights From Statistical Learning Theory

With the above example in mind, let us now consider the problem of pattern recognition in a slightly more formal setting [559, 152, 186]. This will allow us to indicate the factors affecting the design of "better" algorithms. Rather than just

Parzen windows estimators of the two class densities,

$$p_+(x) := \frac{1}{m_+} \sum_{\{i|y_i=+1\}} k(x, x_i) \text{ and } p_-(x) := \frac{1}{m_-} \sum_{\{i|y_i=-1\}} k(x, x_i), \tag{1.14}$$

Parzen Windows

where $x \in \mathcal{X}$.

Given some point x, the label is then simply computed by checking which of the two values $p_+(x)$ or $p_-(x)$ is larger, which leads directly to (1.11). Note that this decision is the best we can do if we have no prior information about the probabilities of the two classes.

The classifier (1.11) is quite close to the type of classifier that this book deals with in detail. Both take the form of kernel expansions on the input domain,

$$y = \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_i k(x, x_i) + b\right). \tag{1.15}$$

In both cases, the expansions correspond to a separating hyperplane in a feature space. In this sense, the α_i can be considered a *dual representation* of the hyperplane's normal vector [223]. Both classifiers are example-based in the sense that the kernels are centered on the training patterns; that is, one of the two arguments of the kernel is always a training pattern. A test point is classified by comparing it to all the training points that appear in (1.15) with a nonzero weight.

More sophisticated classification techniques, to be discussed in the remainder of the book, deviate from (1.11) mainly in the selection of the patterns on which the kernels are centered and in the choice of weights α_i that are placed on the individual kernels in the decision function. It will no longer be the case that *all* training patterns appear in the kernel expansion, and the weights of the kernels in the expansion will no longer be uniform within the classes — recall that in the current example, cf. (1.11), the weights are either $(1/m_+)$ or $(-1/m_-)$, depending on the class to which the pattern belongs.

In the feature space representation, this statement corresponds to saying that we will study normal vectors $\mathbf{w}$ of decision hyperplanes that can be represented as general linear combinations (i.e., with non-uniform coefficients) of the training patterns. For instance, we might want to remove the influence of patterns that are very far away from the decision boundary, either since we expect that they will not improve the generalization error of the decision function, or since we would like to reduce the computational cost of evaluating the decision function (cf. (1.11)). The hyperplane will then only depend on a subset of training patterns called *Support Vectors*.

1.3 Some Insights From Statistical Learning Theory

With the above example in mind, let us now consider the problem of pattern recognition in a slightly more formal setting [559, 152, 186]. This will allow us to indicate the factors affecting the design of "better" algorithms. Rather than just

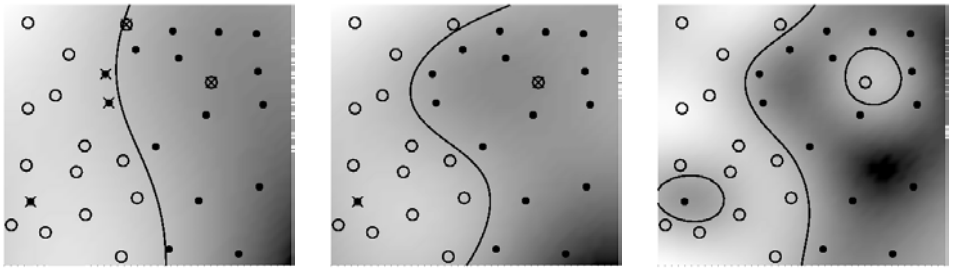


Figure 1.2 2D toy example of binary classification, solved using three models (the decision boundaries are shown). The models vary in complexity, ranging from a simple one *(left)*, which misclassifies a large number of points, to a complex one *(right)*, which "trusts" each point and comes up with solution that is consistent with all training points (but may not work well on new points). As an aside: the plots were generated using the so-called soft-margin SVM to be explained in Chapter 7; cf. also Figure 7.10.

providing tools to come up with new algorithms, we also want to provide some insight in how to do it in a promising way.

In two-class pattern recognition, we seek to infer a function

$$f : \mathcal{X} \to \{\pm 1\} \tag{1.16}$$

from input-output training data (1.1). The training data are sometimes also called the *sample*.

Figure 1.2 shows a simple 2D toy example of a pattern recognition problem. The task is to separate the solid dots from the circles by finding a function which takes the value 1 on the dots and -1 on the circles. Note that instead of plotting this function, we may plot the boundaries where it switches between 1 and -1. In the rightmost plot, we see a classification function which correctly separates all training points. From this picture, however, it is unclear whether the same would hold true for *test* points which stem from the same underlying regularity. For instance, what should happen to a test point which lies close to one of the two "outliers," sitting amidst points of the opposite class? Maybe the outliers should not be allowed to claim their own custom-made regions of the decision function. To avoid this, we could try to go for a simpler model which disregards these points. The leftmost picture shows an almost linear separation of the classes. This separation, however, not only misclassifies the above two outliers, but also a number of "easy" points which are so close to the decision boundary that the classifier really should be able to get them right. Finally, the central picture represents a compromise, by using a model with an intermediate complexity, which gets most points right, without putting too much trust in any individual point.

The goal of statistical learning theory is to place these intuitive arguments in a mathematical framework. To this end, it studies mathematical properties of learning machines. These properties are usually properties of the function class

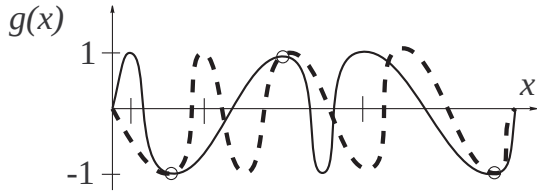


Figure 1.3 A 1D classification problem, with a training set of three points (marked by circles), and three test inputs (marked on the x-axis). Classification is performed by thresholding real-valued functions $g(x)$ according to $\mathrm{sgn}(f(x))$. Note that *both* functions (dotted line, and solid line) perfectly explain the training data, but they give opposite predictions on the test inputs. Lacking any further information, the training data alone give us no means to tell which of the two functions is to be preferred.

that the learning machine can implement.

We assume that the data are generated independently from some unknown (but fixed) probability distribution $P(x, y)$.[5] This is a standard assumption in learning theory; data generated this way is commonly referred to as *iid* (independent and identically distributed). Our goal is to find a function f that will correctly classify unseen examples (x, y), so that $f(x) = y$ for examples (x, y) that are also generated from $P(x, y)$.[6] Correctness of the classification is measured by means of the *zero-one loss function* $c(x, y, f(x)) := \frac{1}{2}|f(x) - y|$. Note that the loss is 0 if (x, y) is classified correctly, and 1 otherwise.

IID Data

Loss Function

If we put no restriction on the set of functions from which we choose our estimated f, however, then even a function that does very well on the training data, e.g., by satisfying $f(x_i) = y_i$ for all $i = 1, \ldots, m$, might not generalize well to unseen examples. To see this, note that for each function f and any test set $(\bar{x}_1, \bar{y}_1), \ldots, (\bar{x}_{\bar{m}}, \bar{y}_{\bar{m}}) \in \mathcal{X} \times \{\pm 1\}$, satisfying $\{\bar{x}_1, \ldots, \bar{x}_{\bar{m}}\} \cap \{x_1, \ldots, x_m\} = \emptyset$, there exists another function f^* such that $f^*(x_i) = f(x_i)$ for all $i = 1, \ldots, m$, yet $f^*(\bar{x}_i) \neq f(\bar{x}_i)$ for all $i = 1, \ldots, \bar{m}$ (cf. Figure 1.3). As we are only given the training data, we have no means of selecting which of the two functions (and hence which of the two different sets of test label predictions) is preferable. We conclude that minimizing only the *(average) training error* (or *empirical risk*),

Test Data

Empirical Risk

$$R_{\mathrm{emp}}[f] = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2}|f(x_i) - y_i|, \tag{1.17}$$

does not imply a small *test error* (called *risk*), averaged over test examples drawn from the underlying distribution $P(x, y)$,

Risk

5. For a definition of a probability distribution, see Section B.1.1.
6. We mostly use the term *example* to denote a pair consisting of a training pattern x and the corresponding target y.

$$R[f] = \int \frac{1}{2}|f(x) - y| \, d\mathrm{P}(x, y). \tag{1.18}$$

The risk can be defined for any loss function, provided the integral exists. For the present zero-one loss function, the risk equals the probability of misclassification.[7]

Capacity

Statistical learning theory (Chapter 5, [570, 559, 561, 136, 562, 14]), or VC (Vapnik-Chervonenkis) theory, shows that it is imperative to restrict the set of functions from which f is chosen to one that has a *capacity* suitable for the amount of available training data. VC theory provides *bounds* on the test error. The minimization of these bounds, which depend on both the empirical risk and the capacity of the function class, leads to the principle of *structural risk minimization* [559].

VC dimension

Shattering

The best-known capacity concept of VC theory is the *VC dimension*, defined as follows: each function of the class separates the patterns in a certain way and thus induces a certain labelling of the patterns. Since the labels are in $\{\pm 1\}$, there are at most 2^m different labellings for m patterns. A very rich function class might be able to realize all 2^m separations, in which case it is said to *shatter* the m points. However, a given class of functions might not be sufficiently righ to shatter the m points. The VC dimension is defined as the largest m such that there exists a set of m points which the class can shatter, and ∞ if no such m exists. It can be thought of as a one-number summary of a learning machine's capacity (for an example, see Figure 1.4). As such, it is necessarily somewhat crude. More accurate capacity measures are the *annealed VC entropy* or the *growth function*. These are usually considered to be harder to evaluate, but they play a fundamental role in the conceptual part of VC theory. Another interesting capacity measure, which can be thought of as a scale-sensitive version of the VC dimension, is the *fat shattering dimension* [286, 6]. For further details, cf. Chapters 5 and 12.

VC Bound

Whilst it will be difficult for the non-expert to appreciate the results of VC theory in this chapter, we will nevertheless briefly describe an example of a VC bound:

7. The risk-based approach to machine learning has its roots in statistical decision theory [582, 166, 43]. In that context, $f(x)$ is thought of as an *action*, and the loss function measures the loss incurred by taking action $f(x)$ upon observing x when the true output (state of nature) is y.

Like many fields of statistics, decision theory comes in two flavors. The present approach is a *frequentist* one. It considers the risk as a function of the distribution P and the decision function f. The *Bayesian* approach considers parametrized families P_Θ to model the distribution. Given a prior over Θ (which need not in general be a finite-dimensional vector), the *Bayes risk* of a decision function f is the *expected* frequentist risk, where the expectation is taken over the prior. Minimizing the Bayes risk (over decision functions) then leads to a *Bayes decision function*. Bayesians thus act as if the parameter Θ were actually a random variable whose distribution is known. Frequentists, who do not make this (somewhat bold) assumption, have to resort to other strategies for picking a decision function. Examples thereof are considerations like *invariance* and *unbiasedness*, both used to restrict the class of decision rules, and the *minimax* principle. A decision function is said to be minimax if it minimizes (over all decision functions) the maximal (over all distributions) risk. For a discussion of the relationship of these issues to VC theory, see Problem 5.9.

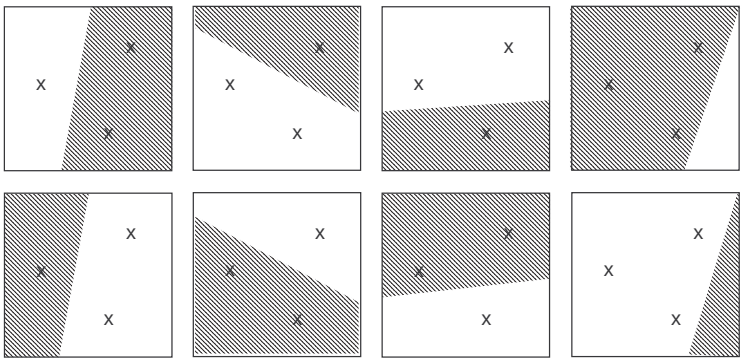


Figure 1.4 A simple VC dimension example. There are $2^3 = 8$ ways of assigning 3 points to two classes. For the displayed points in $\mathbb{R}^2$, all 8 possibilities can be realized using separating hyperplanes, in other words, the function class can shatter 3 points. This would not work if we were given 4 points, no matter how we placed them. Therefore, the VC dimension of the class of separating hyperplanes in $\mathbb{R}^2$ is 3.

if $h < m$ is the VC dimension of the class of functions that the learning machine can implement, then for all functions of that class, independent of the underlying distribution P generating the data, with a probability of at least $1 - \delta$ over the drawing of the training sample,[8] the bound

$$R[f] \leq R_{\text{emp}}[f] + \phi(h, m, \delta) \tag{1.19}$$

holds, where the *confidence term* (or *capacity term*) ϕ is defined as

$$\phi(h, m, \delta) = \sqrt{\frac{1}{m}\left(h\left(\ln\frac{2m}{h} + 1\right) + \ln\frac{4}{\delta}\right)}. \tag{1.20}$$

The bound (1.19) merits further explanation. Suppose we wanted to learn a "dependency" where patterns and labels are statistically independent, $\mathrm{P}(x, y) = \mathrm{P}(x)\mathrm{P}(y)$. In that case, the pattern x contains no information about the label y. If, moreover, the two classes $+1$ and -1 are equally likely, there is no way of making a good guess about the label of a test pattern.

Nevertheless, given a training set of finite size, we can always come up with a learning machine which achieves zero training error (provided we have no examples contradicting each other, i.e., whenever two patterns are identical, then they must come with the same label). To reproduce the random labellings by correctly separating all training examples, however, this machine will necessarily require a large VC dimension h. Therefore, the confidence term (1.20), which increases monotonically with h, will be large, and the bound (1.19) will show

8. Recall that each training example is generated from $P(x, y)$, and thus the training data are subject to randomness.

that the small training error does not guarantee a small test error. This illustrates how the bound can apply independent of assumptions about the underlying distribution $P(x, y)$: it always holds (provided that $h < m$), but it does not always make a nontrivial prediction. In order to get nontrivial predictions from (1.19), the function class must be *restricted* such that its capacity (e.g., VC dimension) is small enough (in relation to the available amount of data). At the same time, the class should be large enough to provide functions that are able to model the dependencies hidden in $P(x, y)$. The choice of the set of functions is thus crucial for learning from data. In the next section, we take a closer look at a class of functions which is particularly interesting for pattern recognition problems.

1.4 Hyperplane Classifiers

In the present section, we shall describe a hyperplane learning algorithm that can be performed in a dot product space (such as the feature space that we introduced earlier). As described in the previous section, to design learning algorithms whose statistical effectiveness can be controlled, one needs to come up with a class of functions whose capacity can be computed. Vapnik et al. [573, 566, 570] considered the class of hyperplanes in some dot product space $\mathcal{H}$,

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = 0 \text{ where } \mathbf{w} \in \mathcal{H}, b \in \mathbb{R}, \tag{1.21}$$

corresponding to decision functions

$$f(\mathbf{x}) = \operatorname{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle + b), \tag{1.22}$$

and proposed a learning algorithm for problems which are separable by hyperplanes (sometimes said to be *linearly separable*), termed the *Generalized Portrait*, for constructing f from empirical data. It is based on two facts. First (see Chapter 7), among all hyperplanes separating the data, there exists a unique *optimal hyperplane*, distinguished by the maximum margin of separation between any training point and the hyperplane. It is the solution of

Optimal Hyperplane

$$\underset{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}}{\text{maximize}} \ \min \{ \|\mathbf{x} - \mathbf{x}_i\| \, | \mathbf{x} \in \mathcal{H}, \langle \mathbf{w}, \mathbf{x} \rangle + b = 0, i = 1, \ldots, m \}. \tag{1.23}$$

Second (see Chapter 5), the capacity (as discussed in Section 1.3) of the class of separating hyperplanes decreases with increasing margin. Hence there are theoretical arguments supporting the good generalization performance of the optimal hyperplane, cf. Chapters 5, 7, 12. In addition, it is *computationally* attractive, since we will show below that it can be constructed by solving a quadratic programming problem for which efficient algorithms exist (see Chapters 6 and 10).

Note that the form of the decision function (1.22) is quite similar to our earlier example (1.9). The ways in which the classifiers are trained, however, are different. In the earlier example, the normal vector of the hyperplane was trivially computed from the class means as $\mathbf{w} = \mathbf{c}_+ - \mathbf{c}_-$.

that the small training error does not guarantee a small test error. This illustrates how the bound can apply independent of assumptions about the underlying distribution $P(x, y)$: it always holds (provided that $h < m$), but it does not always make a nontrivial prediction. In order to get nontrivial predictions from (1.19), the function class must be *restricted* such that its capacity (e.g., VC dimension) is small enough (in relation to the available amount of data). At the same time, the class should be large enough to provide functions that are able to model the dependencies hidden in $P(x, y)$. The choice of the set of functions is thus crucial for learning from data. In the next section, we take a closer look at a class of functions which is particularly interesting for pattern recognition problems.

1.4 Hyperplane Classifiers

In the present section, we shall describe a hyperplane learning algorithm that can be performed in a dot product space (such as the feature space that we introduced earlier). As described in the previous section, to design learning algorithms whose statistical effectiveness can be controlled, one needs to come up with a class of functions whose capacity can be computed. Vapnik et al. [573, 566, 570] considered the class of hyperplanes in some dot product space $\mathcal{H}$,

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = 0 \text{ where } \mathbf{w} \in \mathcal{H}, b \in \mathbb{R}, \tag{1.21}$$

corresponding to decision functions

$$f(\mathbf{x}) = \operatorname{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle + b), \tag{1.22}$$

and proposed a learning algorithm for problems which are separable by hyperplanes (sometimes said to be *linearly separable*), termed the *Generalized Portrait*, for constructing f from empirical data. It is based on two facts. First (see Chapter 7), among all hyperplanes separating the data, there exists a unique *optimal hyperplane*, distinguished by the maximum margin of separation between any training point and the hyperplane. It is the solution of

Optimal Hyperplane

$$\underset{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}}{\text{maximize}} \ \min \{ \|\mathbf{x} - \mathbf{x}_i\| \, | \mathbf{x} \in \mathcal{H}, \langle \mathbf{w}, \mathbf{x} \rangle + b = 0, i = 1, \ldots, m \}. \tag{1.23}$$

Second (see Chapter 5), the capacity (as discussed in Section 1.3) of the class of separating hyperplanes decreases with increasing margin. Hence there are theoretical arguments supporting the good generalization performance of the optimal hyperplane, cf. Chapters 5, 7, 12. In addition, it is *computationally* attractive, since we will show below that it can be constructed by solving a quadratic programming problem for which efficient algorithms exist (see Chapters 6 and 10).

Note that the form of the decision function (1.22) is quite similar to our earlier example (1.9). The ways in which the classifiers are trained, however, are different. In the earlier example, the normal vector of the hyperplane was trivially computed from the class means as $\mathbf{w} = \mathbf{c}_+ - \mathbf{c}_-$.

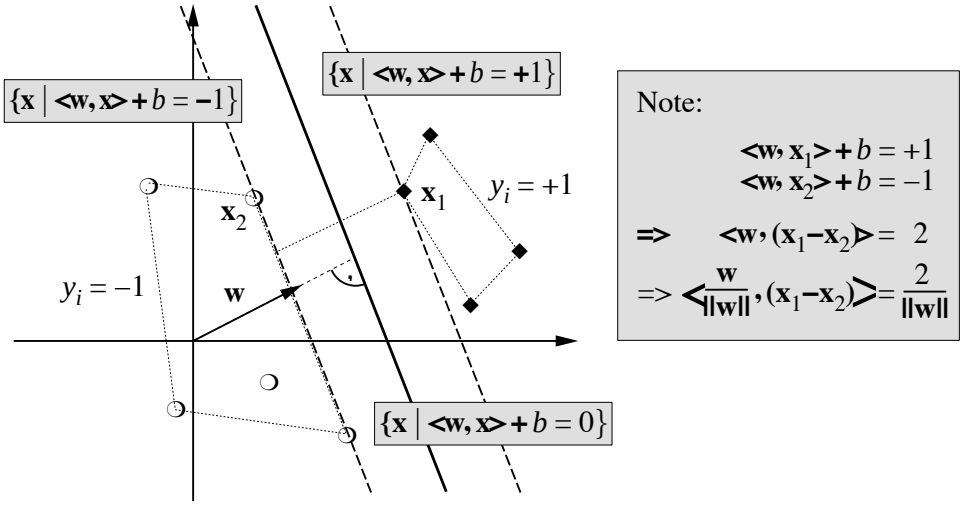


Figure 1.5 A binary classification toy problem: separate balls from diamonds. The *optimal hyperplane* (1.23) is shown as a solid line. The problem being separable, there exists a weight vector $\mathbf{w}$ and a threshold b such that $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 0$ $(i = 1, \ldots, m)$. Rescaling $\mathbf{w}$ and b such that the point(s) closest to the hyperplane satisfy $|\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1$, we obtain a *canonical* form $(\mathbf{w}, b)$ of the hyperplane, satisfying $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$. Note that in this case, the *margin* (the distance of the closest point to the hyperplane) equals $1/\|\mathbf{w}\|$. This can be seen by considering two points $\mathbf{x}_1, \mathbf{x}_2$ on opposite sides of the margin, that is, $\langle \mathbf{w}, \mathbf{x}_1 \rangle + b = 1, \langle \mathbf{w}, \mathbf{x}_2 \rangle + b = -1$, and projecting them onto the hyperplane normal vector $\mathbf{w}/\|\mathbf{w}\|$.

In the present case, we need to do some additional work to find the normal vector that leads to the largest margin. To construct the optimal hyperplane, we have to solve

$$\underset{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}}{\text{minimize}} \quad \tau(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 \tag{1.24}$$

$$\text{subject to} \quad y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \text{ for all } i = 1, \ldots, m. \tag{1.25}$$

Note that the constraints (1.25) ensure that $f(\mathbf{x}_i)$ will be $+1$ for $y_i = +1$, and -1 for $y_i = -1$. Now one might argue that for this to be the case, we don't actually need the "≥ 1" on the right hand side of (1.25). However, without it, it would not be meaningful to minimize the length of $\mathbf{w}$: to see this, imagine we wrote "> 0" instead of "≥ 1." Now assume that the solution is $(\mathbf{w}, b)$. Let us rescale this solution by multiplication with some $0 < \lambda < 1$. Since $\lambda > 0$, the constraints are still satisfied. Since $\lambda < 1$, however, the length of $\mathbf{w}$ has decreased. Hence $(\mathbf{w}, b)$ cannot be the minimizer of $\tau(\mathbf{w})$.

The "≥ 1" on the right hand side of the constraints effectively fixes the scaling of $\mathbf{w}$. In fact, any other positive number would do.

Let us now try to get an intuition for why we should be minimizing the length of $\mathbf{w}$, as in (1.24). If $\|\mathbf{w}\|$ were 1, then the left hand side of (1.25) would equal the distance from $\mathbf{x}_i$ to the hyperplane (cf. (1.23)). In general, we have to divide

$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)$ by $\|\mathbf{w}\|$ to transform it into this distance. Hence, if we can satisfy (1.25) for all $i = 1, \ldots, m$ with an $\mathbf{w}$ of minimal length, then the overall margin will be maximized.

A more detailed explanation of why this leads to the maximum margin hyperplane will be given in Chapter 7. A short summary of the argument is also given in Figure 1.5.

The function τ in (1.24) is called the *objective function*, while (1.25) are called *inequality constraints*. Together, they form a so-called *constrained optimization problem*. Problems of this kind are dealt with by introducing *Lagrange multipliers* $\alpha_i \geq 0$ and a *Lagrangian*[9]

Lagrangian

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i \left(y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1 \right). \tag{1.26}$$

The Lagrangian L has to be minimized with respect to the *primal variables* $\mathbf{w}$ and b and maximized with respect to the *dual variables* α_i (in other words, a saddle point has to be found). Note that the constraint has been incorporated into the second term of the Lagrangian; it is not necessary to enforce it explicitly.

Let us try to get some intuition for this way of dealing with constrained optimization problems. If a constraint (1.25) is violated, then $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1 < 0$, in which case L can be increased by increasing the corresponding α_i. At the same time, $\mathbf{w}$ and b will have to change such that L decreases. To prevent $\alpha_i \left(y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1 \right)$ from becoming an arbitrarily large negative number, the change in $\mathbf{w}$ and b will ensure that, provided the problem is separable, the constraint will eventually be satisfied. Similarly, one can understand that for all constraints which are not precisely met as equalities (that is, for which $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1 > 0$), the corresponding α_i must be 0: this is the value of α_i that maximizes L. The latter is the statement of the Karush-Kuhn-Tucker (KKT) complementarity conditions of optimization theory (Chapter 6).

KKT Conditions

The statement that at the saddle point, the derivatives of L with respect to the primal variables must vanish,

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0 \text{ and } \frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \tag{1.27}$$

leads to

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{1.28}$$

and

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i. \tag{1.29}$$

9. Henceforth, we use boldface Greek letters as a shorthand for corresponding vectors $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$.

Support Vector

The solution vector thus has an expansion (1.29) in terms of a subset of the training patterns, namely those patterns with non-zero α_i, called *Support Vectors (SVs)* (cf. (1.15) in the initial example). By the KKT conditions,

$$\alpha_i \left[y_i \left(\langle \mathbf{x}_i, \mathbf{w} \rangle + b \right) - 1 \right] = 0 \text{ for all } i = 1, \ldots, m, \tag{1.30}$$

the SVs lie on the margin (cf. Figure 1.5). All remaining training examples $(\mathbf{x}_j, y_j)$ are irrelevant: their constraint $y_j(\langle \mathbf{w}, \mathbf{x}_j \rangle + b) \geq 1$ (cf. (1.25)) could just as well be left out, and they do not appear in the expansion (1.29). This nicely captures our intuition of the problem: as the hyperplane (cf. Figure 1.5) is completely determined by the patterns closest to it, the solution should not depend on the other examples.

Dual Problem

By substituting (1.28) and (1.29) into the Lagrangian (1.26), one eliminates the primal variables $\mathbf{w}$ and b, arriving at the so-called *dual optimization problem*, which is the problem that one usually solves in practice:

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximize}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{1.31}$$

$$\text{subject to} \quad \alpha_i \geq 0 \text{ for all } i = 1, \ldots, m \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{1.32}$$

Decision Function

Using (1.29), the hyperplane decision function (1.22) can thus be written as

$$f(\mathbf{x}) = \operatorname{sgn} \left(\sum_{i=1}^{m} y_i \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b \right), \tag{1.33}$$

where b is computed by exploiting (1.30) (for details, cf. Chapter 7).

The structure of the optimization problem closely resembles those that typically arise in Lagrange's formulation of mechanics (e.g., [206]). In the latter class of problem, it is also often the case that only a subset of constraints become active. For instance, if we keep a ball in a box, then it will typically roll into one of the corners. The constraints corresponding to the walls which are not touched by the ball are irrelevant, and those walls could just as well be removed.

Mechanical Analogy

Seen in this light, it is not too surprising that it is possible to give a mechanical interpretation of optimal margin hyperplanes [87]: If we assume that each SV $\mathbf{x}_i$ exerts a perpendicular force of size α_i and direction $y_j \cdot \mathbf{w}/\|\mathbf{w}\|$ on a solid plane sheet lying along the hyperplane, then the solution satisfies the requirements for mechanical stability. The constraint (1.28) states that the forces on the sheet sum to zero, and (1.29) implies that the torques also sum to zero, via $\sum_i \mathbf{x}_i \times y_i \alpha_i \mathbf{w}/\|\mathbf{w}\| = \mathbf{w} \times \mathbf{w}/\|\mathbf{w}\| = 0$.[10] This mechanical analogy illustrates the physical meaning of the term *Support* Vector.

10. Here, the $\times$ denotes the *vector* (or *cross*) *product*, satisfying $\mathbf{v} \times \mathbf{v} = 0$ for all $\mathbf{v} \in \mathcal{H}$.

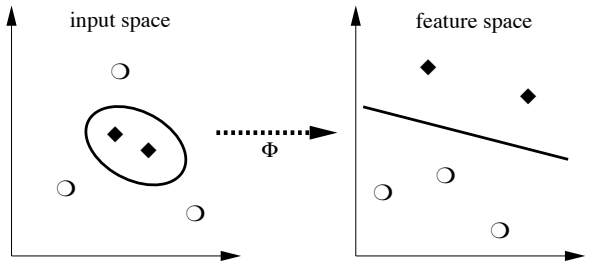


Figure 1.6 The idea of SVMs: map the training data into a higher-dimensional feature space via Φ, and construct a separating hyperplane with maximum margin there. This yields a nonlinear decision boundary in input space. By the use of a kernel function (1.2), it is possible to compute the separating hyperplane without explicitly carrying out the map into the feature space.

1.5 Support Vector Classification

We now have all the tools to describe SVMs (Figure 1.6). Everything in the last section was formulated in a dot product space. We think of this space as the feature space $\mathcal{H}$ of Section 1.1. To express the formulas in terms of the input patterns in $\mathcal{X}$, we thus need to employ (1.6), which expresses the dot product of bold face feature vectors $\mathbf{x}, \mathbf{x}'$ in terms of the kernel k evaluated on input patterns x, x',

$$k(x, x') = \langle \mathbf{x}, \mathbf{x}' \rangle . \tag{1.34}$$

This substitution, which is sometimes referred to as the *kernel trick*, was used by Boser, Guyon, and Vapnik [62] to extend the Generalized Portrait hyperplane classifier to nonlinear Support Vector Machines. Aizerman, Braverman, and Rozonoér [4] called $\mathcal{H}$ the *linearization space*, and used it in the context of the potential function classification method to express the dot product between elements of $\mathcal{H}$ in terms of elements of the input space.

The kernel trick can be applied since all feature vectors only occurred in dot products (see (1.31) and (1.33)). The weight vector (cf. (1.29)) then becomes an expansion in feature space, and therefore will typically no longer correspond to the Φ-image of a single input space vector (cf. Chapter 18). We obtain decision functions of the form (cf. (1.33))

Decision Function

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i \langle \Phi(x), \Phi(x_i) \rangle + b\right) = \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i k(x, x_i) + b\right), \tag{1.35}$$

and the following quadratic program (cf. (1.31)):

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximize}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \tag{1.36}$$

$$\text{subject to} \quad \alpha_i \geq 0 \text{ for all } i = 1, \ldots, m, \quad \text{and} \quad \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{1.37}$$

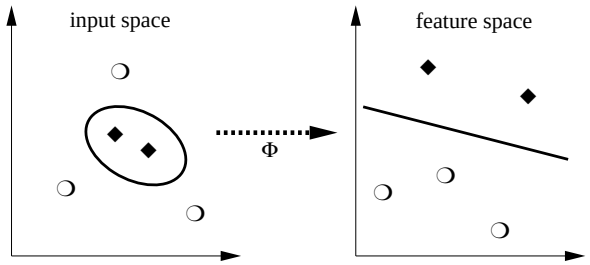


Figure 1.6 The idea of SVMs: map the training data into a higher-dimensional feature space via Φ, and construct a separating hyperplane with maximum margin there. This yields a nonlinear decision boundary in input space. By the use of a kernel function (1.2), it is possible to compute the separating hyperplane without explicitly carrying out the map into the feature space.

1.5 Support Vector Classification

We now have all the tools to describe SVMs (Figure 1.6). Everything in the last section was formulated in a dot product space. We think of this space as the feature space $\mathcal{H}$ of Section 1.1. To express the formulas in terms of the input patterns in $\mathcal{X}$, we thus need to employ (1.6), which expresses the dot product of bold face feature vectors $\mathbf{x}, \mathbf{x}'$ in terms of the kernel k evaluated on input patterns x, x',

$$k(x, x') = \langle \mathbf{x}, \mathbf{x}' \rangle . \tag{1.34}$$

This substitution, which is sometimes referred to as the *kernel trick*, was used by Boser, Guyon, and Vapnik [62] to extend the Generalized Portrait hyperplane classifier to nonlinear Support Vector Machines. Aizerman, Braverman, and Rozonoér [4] called $\mathcal{H}$ the *linearization space*, and used it in the context of the potential function classification method to express the dot product between elements of $\mathcal{H}$ in terms of elements of the input space.

The kernel trick can be applied since all feature vectors only occurred in dot products (see (1.31) and (1.33)). The weight vector (cf. (1.29)) then becomes an expansion in feature space, and therefore will typically no longer correspond to the Φ-image of a single input space vector (cf. Chapter 18). We obtain decision functions of the form (cf. (1.33))

Decision Function

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i \langle \Phi(x), \Phi(x_i) \rangle + b\right) = \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i k(x, x_i) + b\right), \tag{1.35}$$

and the following quadratic program (cf. (1.31)):

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximize}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \tag{1.36}$$

$$\text{subject to} \quad \alpha_i \geq 0 \text{ for all } i = 1, \ldots, m, \quad \text{and} \quad \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{1.37}$$

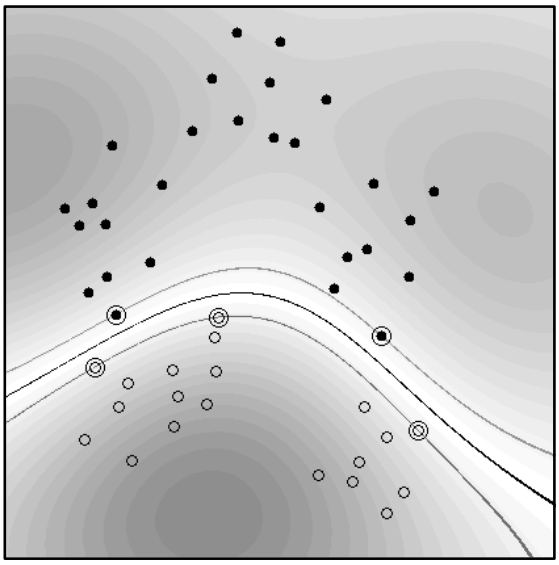


Figure 1.7 Example of an SV classifier found using a radial basis function kernel $k(x, x') = \exp(-\|x - x'\|^2)$ (here, the input space is $\mathcal{X} = [-1, 1]^2$). Circles and disks are two classes of training examples; the middle line is the decision surface; the outer lines precisely meet the constraint (1.25). Note that the SVs found by the algorithm (marked by extra circles) are not centers of clusters, but examples which are critical for the given classification task. Gray values code $|\sum_{i=1}^{m} y_i \alpha_i k(x, x_i) + b|$, the modulus of the argument of the decision function (1.35). The top and the bottom lines indicate places where it takes the value 1 (from [471]).

Figure 1.7 shows an example of this approach, using a Gaussian radial basis function kernel. We will later study the different possibilities for the kernel function in detail (Chapters 2 and 13).

Soft Margin Hyperplane

In practice, a separating hyperplane may not exist, e.g., if a high noise level causes a large overlap of the classes. To allow for the possibility of examples violating (1.25), one introduces slack variables [111, 561, 481]

$$\xi_i \geq 0 \text{ for all } i = 1, \ldots, m, \tag{1.38}$$

in order to relax the constraints (1.25) to

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i \text{ for all } i = 1, \ldots, m. \tag{1.39}$$

A classifier that generalizes well is then found by controlling both the classifier capacity (via $\|\mathbf{w}\|$) and the sum of the slacks $\sum_i \xi_i$. The latter can be shown to provide an upper bound on the number of training errors.

One possible realization of such a *soft margin* classifier is obtained by minimizing the objective function

$$\tau(\mathbf{w}, \boldsymbol{\xi}) = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \xi_i \tag{1.40}$$

subject to the constraints (1.38) and (1.39), where the constant $C > 0$ determines the trade-off between margin maximization and training error minimization.[11] Incorporating a kernel, and rewriting it in terms of Lagrange multipliers, this again leads to the problem of maximizing (1.36), subject to the constraints

$$0 \leq \alpha_i \leq C \text{ for all } i = 1, \ldots, m, \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{1.41}$$

The only difference from the separable case is the upper bound C on the Lagrange multipliers α_i. This way, the influence of the individual patterns (which could be outliers) gets limited. As above, the solution takes the form (1.35). The threshold b can be computed by exploiting the fact that for all SVs x_i with $\alpha_i < C$, the slack variable ξ_i is zero (this again follows from the KKT conditions), and hence

$$\sum_{j=1}^{m} \alpha_j y_j k(x_i, x_j) + b = y_i. \tag{1.42}$$

Geometrically speaking, choosing b amounts to shifting the hyperplane, and (1.42) states that we have to shift the hyperplane such that the SVs with zero slack variables lie on the ± 1 lines of Figure 1.5.

Another possible realization of a soft margin variant of the optimal hyperplane uses the more natural ν-parametrization. In it, the parameter C is replaced by a parameter $\nu \in (0, 1]$ which can be shown to provide lower and upper bounds for the fraction of examples that will be SVs and those that will have non-zero slack variables, respectively. It uses a primal objective function with the error term $\left(\frac{1}{\nu m} \sum_i \xi_i\right) - \rho$ instead of $C \sum_i \xi_i$ (cf. (1.40)), and separation constraints that involve a margin parameter ρ,

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq \rho - \xi_i \text{ for all } i = 1, \ldots, m, \tag{1.43}$$

which itself is a variable of the optimization problem. The dual can be shown to consist in maximizing the quadratic part of (1.36), subject to $0 \leq \alpha_i \leq 1/(\nu m)$, $\sum_i \alpha_i y_i = 0$ and the additional constraint $\sum_i \alpha_i = 1$. We shall return to these methods in more detail in Section 7.5.

1.6 Support Vector Regression

Let us turn to a problem slightly more general than pattern recognition. Rather than dealing with outputs $y \in \{\pm 1\}$, *regression estimation* is concerned with estimating real-valued functions.

To generalize the SV algorithm to the regression case, an analog of the soft margin is constructed in the space of the target values y (note that we now have

11. It is sometimes convenient to scale the sum in (1.40) by C/m rather than C, as done in Chapter 7 below.

subject to the constraints (1.38) and (1.39), where the constant $C > 0$ determines the trade-off between margin maximization and training error minimization.[11] Incorporating a kernel, and rewriting it in terms of Lagrange multipliers, this again leads to the problem of maximizing (1.36), subject to the constraints

$$0 \leq \alpha_i \leq C \text{ for all } i = 1, \ldots, m, \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{1.41}$$

The only difference from the separable case is the upper bound C on the Lagrange multipliers α_i. This way, the influence of the individual patterns (which could be outliers) gets limited. As above, the solution takes the form (1.35). The threshold b can be computed by exploiting the fact that for all SVs x_i with $\alpha_i < C$, the slack variable ξ_i is zero (this again follows from the KKT conditions), and hence

$$\sum_{j=1}^{m} \alpha_j y_j k(x_i, x_j) + b = y_i. \tag{1.42}$$

Geometrically speaking, choosing b amounts to shifting the hyperplane, and (1.42) states that we have to shift the hyperplane such that the SVs with zero slack variables lie on the ± 1 lines of Figure 1.5.

Another possible realization of a soft margin variant of the optimal hyperplane uses the more natural ν-parametrization. In it, the parameter C is replaced by a parameter $\nu \in (0, 1]$ which can be shown to provide lower and upper bounds for the fraction of examples that will be SVs and those that will have non-zero slack variables, respectively. It uses a primal objective function with the error term $\left(\frac{1}{\nu m} \sum_i \xi_i\right) - \rho$ instead of $C \sum_i \xi_i$ (cf. (1.40)), and separation constraints that involve a margin parameter ρ,

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq \rho - \xi_i \text{ for all } i = 1, \ldots, m, \tag{1.43}$$

which itself is a variable of the optimization problem. The dual can be shown to consist in maximizing the quadratic part of (1.36), subject to $0 \leq \alpha_i \leq 1/(\nu m)$, $\sum_i \alpha_i y_i = 0$ and the additional constraint $\sum_i \alpha_i = 1$. We shall return to these methods in more detail in Section 7.5.

1.6 Support Vector Regression

Let us turn to a problem slightly more general than pattern recognition. Rather than dealing with outputs $y \in \{\pm 1\}$, *regression estimation* is concerned with estimating real-valued functions.

To generalize the SV algorithm to the regression case, an analog of the soft margin is constructed in the space of the target values y (note that we now have

11. It is sometimes convenient to scale the sum in (1.40) by C/m rather than C, as done in Chapter 7 below.

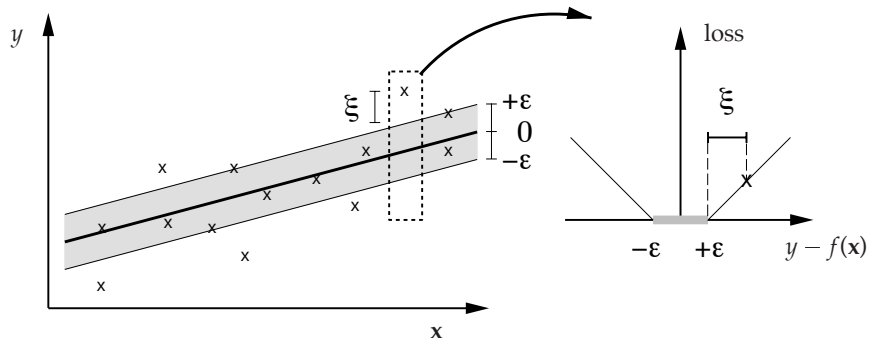


Figure 1.8 In SV regression, a tube with radius ε is fitted to the data. The trade-off between model complexity and points lying outside of the tube (with positive slack variables ξ) is determined by minimizing (1.47).

ε-Insensitive Loss

$y \in \mathbb{R}$) by using Vapnik's *ε-insensitive loss function* [561] (Figure 1.8, see Chapters 3 and 9) . This quantifies the loss incurred by predicting $f(\mathbf{x})$ instead of y as

$$c(x, y, f(x)) := |y - f(\mathbf{x})|_\varepsilon := \max\{0, |y - f(\mathbf{x})| - \varepsilon\}. \tag{1.44}$$

To estimate a linear regression

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b, \tag{1.45}$$

one minimizes

$$\frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} |y_i - f(\mathbf{x}_i)|_\varepsilon. \tag{1.46}$$

Note that the term $\|\mathbf{w}\|^2$ is the same as in pattern recognition (cf. (1.40)); for further details, cf. Chapter 9.

We can transform this into a constrained optimization problem by introducing slack variables, akin to the soft margin case. In the present case, we need two types of slack variable for the two cases $f(\mathbf{x}_i) - y_i > \varepsilon$ and $y_i - f(\mathbf{x}_i) > \varepsilon$. We denote them by $\boldsymbol{\xi}$ and $\boldsymbol{\xi}^*$, respectively, and collectively refer to them as $\boldsymbol{\xi}^{(*)}$.

The optimization problem is given by

$$\underset{\mathbf{w} \in \mathcal{H}, \boldsymbol{\xi}^{(*)} \in \mathbb{R}^m, b \in \mathbb{R}}{\text{minimize}} \quad \tau(\mathbf{w}, \boldsymbol{\xi}^{(*)}) = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} (\xi_i + \xi_i^*) \tag{1.47}$$

$$\text{subject to} \quad f(\mathbf{x}_i) - y_i \leq \varepsilon + \xi_i \tag{1.48}$$

$$y_i - f(\mathbf{x}_i) \leq \varepsilon + \xi_i^* \tag{1.49}$$

$$\xi_i, \xi_i^* \geq 0 \qquad \text{for all } i = 1, \ldots, m. \tag{1.50}$$

Note that according to (1.48) and (1.49), any error smaller than ε does not require a nonzero ξ_i or ξ_i^* and hence does not enter the objective function (1.47).

Generalization to *kernel*-based regression estimation is carried out in an analo-

gous manner to the case of pattern recognition. Introducing Lagrange multipliers, one arrives at the following optimization problem (for $C, \varepsilon \geq 0$ chosen a priori):

$$\begin{aligned} \underset{\boldsymbol{\alpha}, \boldsymbol{\alpha}^* \in \mathbb{R}^m}{\text{maximize}} \quad W(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*) \quad = \quad & -\varepsilon \sum_{i=1}^{m} (\alpha_i^* + \alpha_i) + \sum_{i=1}^{m} (\alpha_i^* - \alpha_i) y_i \\ & -\frac{1}{2} \sum_{i,j=1}^{m} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) k(x_i, x_j) \end{aligned} \tag{1.51}$$

$$\text{subject to} \quad 0 \leq \alpha_i, \alpha_i^* \leq C \text{ for all } i = 1, \ldots, m, \text{ and } \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) = 0. \tag{1.52}$$

Regression Function

The regression estimate takes the form

$$f(x) = \sum_{i=1}^{m} (\alpha_i^* - \alpha_i) k(x_i, x) + b, \tag{1.53}$$

where b is computed using the fact that (1.48) becomes an equality with $\xi_i = 0$ if $0 < \alpha_i < C$, and (1.49) becomes an equality with $\xi_i^* = 0$ if $0 < \alpha_i^* < C$ (for details, see Chapter 9). The solution thus looks quite similar to the pattern recognition case (cf. (1.35) and Figure 1.9).

A number of extensions of this algorithm are possible. From an abstract point of view, we just need some target function which depends on $(\mathbf{w}, \boldsymbol{\xi})$ (cf. (1.47)). There are multiple degrees of freedom for constructing it, including some freedom how to penalize, or regularize. For instance, more general loss functions can be used for $\boldsymbol{\xi}$, leading to problems that can still be solved efficiently ([512, 515], cf. Chapter 9). Moreover, norms other than the 2-norm $\|.\|$ can be used to regularize the solution (see Sections 4.9 and 9.4).

Finally, the algorithm can be modified such that ε need not be specified a priori. Instead, one specifies an upper bound $0 \leq \nu \leq 1$ on the fraction of points allowed to lie outside the tube (asymptotically, the number of SVs) and the corresponding ε is computed automatically. This is achieved by using as primal objective function

ν-SV Regression

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\left(\nu m \varepsilon + \sum_{i=1}^{m} |y_i - f(\mathbf{x}_i)|_\varepsilon\right) \tag{1.54}$$

instead of (1.46), and treating $\varepsilon \geq 0$ as a parameter over which we minimize. For more detail, cf. Section 9.3.

1.7 Kernel Principal Component Analysis

The kernel method for computing dot products in feature spaces is not restricted to SVMs. Indeed, it has been pointed out that it can be used to develop nonlinear generalizations of any algorithm that can be cast in terms of dot products, such as principal component analysis (PCA) [480].

Principal component analysis is perhaps the most common feature extraction algorithm; for details, see Chapter 14. The term *feature extraction* commonly refers

gous manner to the case of pattern recognition. Introducing Lagrange multipliers, one arrives at the following optimization problem (for $C, \varepsilon \geq 0$ chosen a priori):

$$\begin{aligned} \underset{\boldsymbol{\alpha}, \boldsymbol{\alpha}^* \in \mathbb{R}^m}{\text{maximize}} \quad W(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*) \quad = \quad & -\varepsilon \sum_{i=1}^{m} (\alpha_i^* + \alpha_i) + \sum_{i=1}^{m} (\alpha_i^* - \alpha_i) y_i \\ & -\frac{1}{2} \sum_{i,j=1}^{m} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) k(x_i, x_j) \end{aligned} \tag{1.51}$$

$$\text{subject to} \quad 0 \leq \alpha_i, \alpha_i^* \leq C \text{ for all } i = 1, \ldots, m, \text{ and } \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) = 0. \tag{1.52}$$

Regression Function

The regression estimate takes the form

$$f(x) = \sum_{i=1}^{m} (\alpha_i^* - \alpha_i) k(x_i, x) + b, \tag{1.53}$$

where b is computed using the fact that (1.48) becomes an equality with $\xi_i = 0$ if $0 < \alpha_i < C$, and (1.49) becomes an equality with $\xi_i^* = 0$ if $0 < \alpha_i^* < C$ (for details, see Chapter 9). The solution thus looks quite similar to the pattern recognition case (cf. (1.35) and Figure 1.9).

A number of extensions of this algorithm are possible. From an abstract point of view, we just need some target function which depends on $(\mathbf{w}, \boldsymbol{\xi})$ (cf. (1.47)). There are multiple degrees of freedom for constructing it, including some freedom how to penalize, or regularize. For instance, more general loss functions can be used for $\boldsymbol{\xi}$, leading to problems that can still be solved efficiently ([512, 515], cf. Chapter 9). Moreover, norms other than the 2-norm $\|.\|$ can be used to regularize the solution (see Sections 4.9 and 9.4).

Finally, the algorithm can be modified such that ε need not be specified a priori. Instead, one specifies an upper bound $0 \leq \nu \leq 1$ on the fraction of points allowed to lie outside the tube (asymptotically, the number of SVs) and the corresponding ε is computed automatically. This is achieved by using as primal objective function

ν-SV Regression

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\left(\nu m \varepsilon + \sum_{i=1}^{m} |y_i - f(\mathbf{x}_i)|_\varepsilon\right) \tag{1.54}$$

instead of (1.46), and treating $\varepsilon \geq 0$ as a parameter over which we minimize. For more detail, cf. Section 9.3.

1.7 Kernel Principal Component Analysis

The kernel method for computing dot products in feature spaces is not restricted to SVMs. Indeed, it has been pointed out that it can be used to develop nonlinear generalizations of any algorithm that can be cast in terms of dot products, such as principal component analysis (PCA) [480].

Principal component analysis is perhaps the most common feature extraction algorithm; for details, see Chapter 14. The term *feature extraction* commonly refers

to procedures for extracting (real) numbers from patterns which in some sense represent the crucial information contained in these patterns.

PCA in feature space leads to an algorithm called *kernel PCA*. By solving an eigenvalue problem, the algorithm computes nonlinear feature extraction functions

$$f_n(x) = \sum_{i=1}^{m} \alpha_i^n k(x_i, x), \tag{1.55}$$

where, up to a normalizing constant, the α_i^n are the components of the nth eigenvector of the kernel matrix $K_{ij} := (k(x_i, x_j))$.

In a nutshell, this can be understood as follows. To do PCA in $\mathcal{H}$, we wish to find eigenvectors $\mathbf{v}$ and eigenvalues λ of the so-called *covariance matrix* $\mathbf{C}$ in the feature space, where

$$\mathbf{C} := \frac{1}{m} \sum_{i=1}^{m} \Phi(x_i)\Phi(x_i)^\top. \tag{1.56}$$

Here, $\Phi(x_i)^\top$ denotes the transpose of $\Phi(x_i)$ (see Section B.2.1). In the case when $\mathcal{H}$ is very high dimensional, the computational costs of doing this directly are prohibitive. Fortunately, one can show that all solutions to

$$\mathbf{C}\mathbf{v} = \lambda \mathbf{v} \tag{1.57}$$

with $\lambda \neq 0$ must lie in the span of Φ-images of the training data. Thus, we may expand the solution $\mathbf{v}$ as

$$\mathbf{v} = \sum_{i=1}^{m} \alpha_i \Phi(x_i), \tag{1.58}$$

thereby reducing the problem to that of finding the α_i. It turns out that this leads to a dual eigenvalue problem for the expansion coefficients,

Kernel PCA Eigenvalue Problem

$$m\lambda\boldsymbol{\alpha} = K\boldsymbol{\alpha}, \tag{1.59}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)^\top$.

To extract nonlinear features from a test point x, we compute the dot product between $\Phi(x)$ and the nth normalized eigenvector in feature space,

Feature Extraction

$$\langle \mathbf{v}^n, \Phi(x) \rangle = \sum_{i=1}^{m} \alpha_i^n k(x_i, x). \tag{1.60}$$

Usually, this will be computationally far less expensive than taking the dot product in the feature space explicitly.

A toy example is given in Chapter 14 (Figure 14.4). As in the case of SVMs, the architecture can be visualized by Figure 1.9.

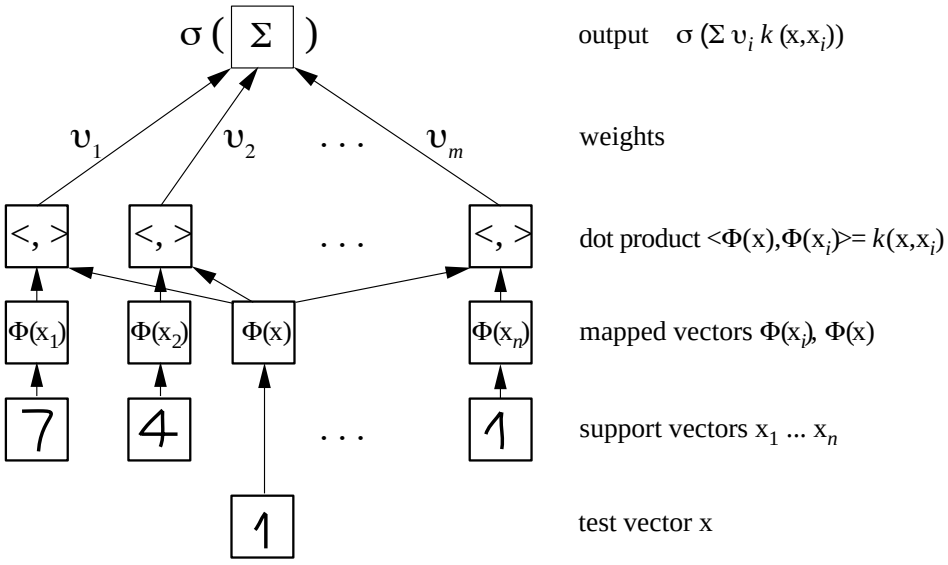


Figure 1.9 Architecture of SVMs and related kernel methods. The input x and the expansion patterns (SVs) x_i (we assume that we are dealing with handwritten digits) are nonlinearly mapped (by Φ) into a feature space $\mathcal{H}$ where dot products are computed. Through the use of the kernel k, these two layers are in practice computed in one step. The results are linearly combined using weights v_i, found by solving a quadratic program (in pattern recognition, $v_i = y_i \alpha_i$; in regression estimation, $v_i = \alpha_i^* - \alpha_i$) or an eigenvalue problem (Kernel PCA). The linear combination is fed into the function σ (in pattern recognition, $\sigma(x) = \operatorname{sgn}(x + b)$; in regression estimation, $\sigma(x) = x + b$; in Kernel PCA, $\sigma(x) = x$).

1.8 Empirical Results and Implementations

Examples of Kernels

Having described the basics of SVMs, we now summarize some empirical findings. By the use of kernels, the optimal margin classifier was turned into a high-performance classifier. Surprisingly, it was observed that the polynomial kernel

$$k(x, x') = \langle x, x' \rangle^d, \tag{1.61}$$

the Gaussian

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\,\sigma^2}\right), \tag{1.62}$$

and the sigmoid

$$k(x, x') = \tanh\left(\kappa\,\langle x, x' \rangle + \Theta\right), \tag{1.63}$$

with suitable choices of $d \in \mathbb{N}$ and $\sigma, \kappa, \Theta \in \mathbb{R}$ (here, $\mathcal{X} \subset \mathbb{R}^N$), empirically led to SV classifiers with very similar accuracies and SV sets (Section 7.8.2). In this sense, the SV set seems to characterize (or *compress*) the given task in a manner which

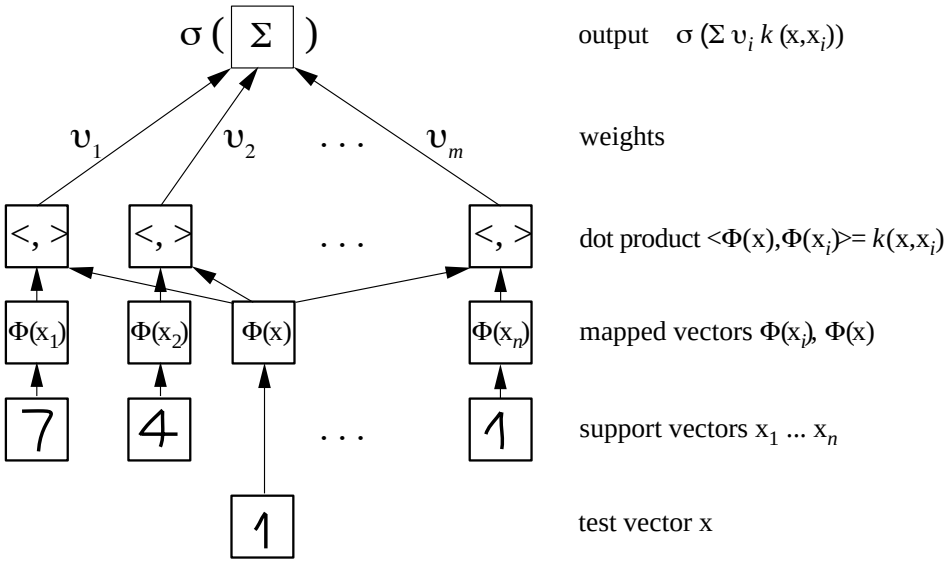


Figure 1.9 Architecture of SVMs and related kernel methods. The input x and the expansion patterns (SVs) x_i (we assume that we are dealing with handwritten digits) are nonlinearly mapped (by Φ) into a feature space $\mathcal{H}$ where dot products are computed. Through the use of the kernel k, these two layers are in practice computed in one step. The results are linearly combined using weights v_i, found by solving a quadratic program (in pattern recognition, $v_i = y_i \alpha_i$; in regression estimation, $v_i = \alpha_i^* - \alpha_i$) or an eigenvalue problem (Kernel PCA). The linear combination is fed into the function σ (in pattern recognition, $\sigma(x) = \operatorname{sgn}(x + b)$; in regression estimation, $\sigma(x) = x + b$; in Kernel PCA, $\sigma(x) = x$).

1.8 Empirical Results and Implementations

Examples of Kernels

Having described the basics of SVMs, we now summarize some empirical findings. By the use of kernels, the optimal margin classifier was turned into a high-performance classifier. Surprisingly, it was observed that the polynomial kernel

$$k(x, x') = \langle x, x' \rangle^d, \tag{1.61}$$

the Gaussian

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\,\sigma^2}\right), \tag{1.62}$$

and the sigmoid

$$k(x, x') = \tanh\left(\kappa\,\langle x, x' \rangle + \Theta\right), \tag{1.63}$$

with suitable choices of $d \in \mathbb{N}$ and $\sigma, \kappa, \Theta \in \mathbb{R}$ (here, $\mathcal{X} \subset \mathbb{R}^N$), empirically led to SV classifiers with very similar accuracies and SV sets (Section 7.8.2). In this sense, the SV set seems to characterize (or *compress*) the given task in a manner which

to some extent is independent of the type of kernel (that is, the type of classifier) used, provided the kernel parameters are well adjusted.

Applications

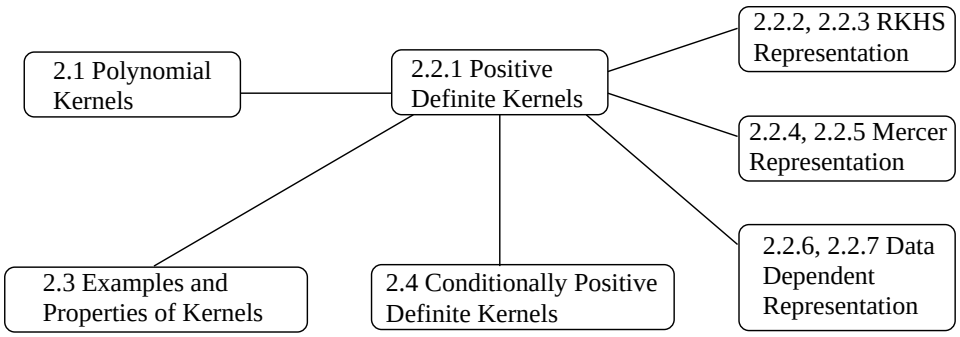
Initial work at AT&T Bell Labs focused on OCR (optical character recognition), a problem where the two main issues are classification accuracy and classification speed. Consequently, some effort went into the improvement of SVMs on these issues, leading to the *Virtual SV* method for incorporating prior knowledge about transformation invariances by transforming SVs (Chapter 7), and the *Reduced Set* method (Chapter 18) for speeding up classification. Using these procedures, SVMs soon became competitive with the best available classifiers on OCR and other object recognition tasks [87, 57, 419, 438, 134], and later even achieved the world record on the main handwritten digit benchmark dataset [134].

Implementation

An initial weakness of SVMs, less apparent in OCR applications which are characterized by low noise levels, was that the size of the quadratic programming problem (Chapter 10) scaled with the number of support vectors. This was due to the fact that in (1.36), the quadratic part contained at least all SVs — the common practice was to extract the SVs by going through the training data in chunks while regularly testing for the possibility that patterns initially not identified as SVs become SVs at a later stage. This procedure is referred to as *chunking*; note that without chunking, the size of the matrix in the quadratic part of the objective function would be $m \times m$, where m is the number of all training examples.

What happens if we have a high-noise problem? In this case, many of the slack variables ξ_i become nonzero, and all the corresponding examples become SVs. For this case, decomposition algorithms were proposed [398, 409], based on the observation that not only can we leave out the non-SV examples (the x_i with $\alpha_i = 0$) from the current chunk, but also some of the SVs, especially those that hit the upper boundary ($\alpha_i = C$). The chunks are usually dealt with using quadratic optimizers. Among the optimizers used for SVMs are LOQO [555], MINOS [380], and variants of conjugate gradient descent, such as the optimizers of Bottou [459] and Burges [85]. Several public domain SV packages and optimizers are listed on the web page http://www.kernel-machines.org. For more details on implementations, see Chapter 10.

Once the SV algorithm had been generalized to regression, researchers started applying it to various problems of estimating real-valued functions. Very good results were obtained on the Boston housing benchmark [529], and on problems of times series prediction (see [376, 371, 351]). Moreover, the SV method was applied to the solution of inverse function estimation problems ([572]; cf. [563, 589]). For overviews, the interested reader is referred to [85, 472, 504, 125].



2.1 Product Features

In this section, we think of $\mathcal{X}$ as a subset of the vector space $\mathbb{R}^N$, ($N \in \mathbb{N}$), endowed with the canonical dot product (1.3).

Monomial Features

Suppose we are given patterns $x \in \mathcal{X}$ where most information is contained in the dth order products (so-called monomials) of entries $[x]_j$ of x,

$$[x]_{j_1} \cdot [x]_{j_2} \cdots [x]_{j_d}, \tag{2.3}$$

where $j_1, \ldots, j_d \in \{1, \ldots, N\}$. Often, these monomials are referred to as *product features*. These features form the basis of many practical algorithms; indeed, there is a whole field of pattern recognition research studying *polynomial classifiers* [484], which is based on first extracting product features and then applying learning algorithms to these features. In other words, the patterns are preprocessed by mapping into the feature space $\mathcal{H}$ of all products of d entries. This has proven quite effective in visual pattern recognition tasks, for instance. To understand the rationale for doing this, note that visual patterns are usually represented as vectors whose entries are the pixel intensities. Taking products of entries of these vectors then corresponds to taking products of pixel intensities, and is thus akin to taking logical "and" operations on the pixels. Roughly speaking, this corresponds to the intuition that, for instance, a handwritten "8" constitutes an eight if there is a top circle *and* a bottom circle. With just one of the two circles, it is not half an "8," but rather a "0." Nonlinearities of this type are crucial for achieving high accuracies in pattern recognition tasks.

Let us take a look at this feature map in the simple example of two-dimensional patterns, for which $\mathcal{X} = \mathbb{R}^2$. In this case, we can collect all monomial feature extractors of degree 2 in the nonlinear map

$$\Phi : \mathbb{R}^2 \to \mathcal{H} = \mathbb{R}^3, \tag{2.4}$$

$$([x]_1, [x]_2) \mapsto ([x]_1^2, [x]_2^2, [x]_1[x]_2). \tag{2.5}$$

This approach works fine for small toy examples, but it fails for realistically sized

problems: for N-dimensional input patterns, there exist

$$N_{\mathcal{H}} = \binom{d+N-1}{d} = \frac{(d+N-1)!}{d!(N-1)!} \tag{2.6}$$

different monomials (2.3) of degree d, comprising a feature space $\mathcal{H}$ of dimension $N_{\mathcal{H}}$. For instance, 16×16 pixel input images and a monomial degree $d = 5$ thus yield a dimension of almost 10^{10}.

In certain cases described below, however, there exists a way of *computing dot products* in these high-dimensional feature spaces without explicitly mapping into the spaces, by means of kernels nonlinear in the input space $\mathbb{R}^N$. Thus, if the subsequent processing can be carried out using dot products exclusively, we are able to deal with the high dimension.

We now describe how dot products in polynomial feature spaces can be computed efficiently, followed by a section in which we discuss more general feature spaces. In order to compute dot products of the form $\langle \Phi(x), \Phi(x') \rangle$, we employ kernel representations of the form

Kernel

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle , \tag{2.7}$$

which allow us to compute the value of the dot product in $\mathcal{H}$ without having to explicitly compute the map Φ.

What does k look like in the case of polynomial features? We start by giving an example for $N = d = 2$, as considered above [561]. For the map

$$\Phi : ([x]_1, [x]_2) \mapsto ([x]_1^2, [x]_2^2, [x]_1[x]_2, [x]_2[x]_1), \tag{2.8}$$

(note that for now, we have considered $[x]_1[x]_2$ and $[x]_2[x]_1$ as separate features; thus we are looking at *ordered* monomials) dot products in $\mathcal{H}$ take the form

$$\langle \Phi(x), \Phi(x') \rangle = [x]_1^2[x']_1^2 + [x]_2^2[x']_2^2 + 2[x]_1[x]_2[x']_1[x']_2 = \langle x, x' \rangle^2 . \tag{2.9}$$

In other words, the desired kernel k is simply the square of the dot product in input space. The same works for arbitrary $N, d \in \mathbb{N}$ [62]: as a straightforward generalization of a result proved in the context of polynomial approximation [412, Lemma 2.1], we have:

Proposition 2.1 *Define C_d to map $x \in \mathbb{R}^N$ to the vector $C_d(x)$ whose entries are all possible dth degree ordered products of the entries of x. Then the corresponding kernel computing the dot product of vectors mapped by C_d is*

$$k(x, x') = \langle C_d(x), C_d(x') \rangle = \langle x, x' \rangle^d . \tag{2.10}$$

Polynomial Kernel

Proof We directly compute

$$\langle C_d(x), C_d(x') \rangle = \sum_{j_1=1}^{N} \dots \sum_{j_d=1}^{N} [x]_{j_1} \cdot \dots \cdot [x]_{j_d} \cdot [x']_{j_1} \cdot \dots \cdot [x']_{j_d} \tag{2.11}$$

$$= \sum_{j_1=1}^{N} [x]_{j_1} \cdot [x']_{j_1} \ldots \sum_{j_d=1}^{N} [x]_{j_d} \cdot [x']_{j_d} = \left(\sum_{j=1}^{N} [x]_j \cdot [x']_j \right)^d = \langle x, x' \rangle^d . \quad \blacksquare$$

Note that we used the symbol C_d for the feature map. The reason for this is that we would like to reserve Φ_d for the corresponding map computing *unordered* product features. Let us construct such a map Φ_d, yielding the same value of the dot product. To this end, we have to compensate for the multiple occurrence of certain monomials in C_d by scaling the respective entries of Φ_d with the square roots of their numbers of occurrence. Then, by this construction of Φ_d, and (2.10),

$$\langle \Phi_d(x), \Phi_d(x') \rangle = \langle C_d(x), C_d(x') \rangle = \langle x, x' \rangle^d . \tag{2.12}$$

For instance, if n of the j_i in (2.3) are equal, and the remaining ones are different, then the coefficient in the corresponding component of Φ_d is $\sqrt{(d-n+1)!}$. For the general case, see Problem 2.2. For Φ_2, this simply means that [561]

$$\Phi_2(x) = ([x]_1^2, [x]_2^2, \sqrt{2}\,[x]_1[x]_2). \tag{2.13}$$

The above reasoning illustrates an important point pertaining to the construction of feature spaces associated with kernel functions. Although they map into different feature spaces, Φ_d and C_d are both valid instantiations of feature maps for $k(x, x') = \langle x, x' \rangle^d$.

To illustrate how monomial feature kernels can significantly simplify pattern recognition tasks, let us consider a simple toy example.

Toy Example

Example 2.2 (Monomial Features in 2-D Pattern Recognition) *In the example of Figure 2.1, a non-separable problem is reduced to the construction of a separating hyperplane by preprocessing the input data with Φ_2. As we shall see in later chapters, this has advantages both from the* computational *point of view (there exist efficient algorithms for computing the hyperplane) and from the* statistical *point of view (there exist guarantees for how well the hyperplane will generalize to unseen test points).*

In more realistic cases, e.g., if x represents an image with the entries being pixel values, polynomial kernels $\langle x, x' \rangle^d$ enable us to work in the space spanned by products of any d pixel values — provided that we are able to do our work solely in terms of dot products, without any explicit usage of a mapped pattern $\Phi_d(x)$. Using kernels of the form (2.10), we can take higher-order statistics into account, without the combinatorial explosion (2.6) of time and memory complexity which accompanies even moderately high N and d.

To conclude this section, note that it is possible to modify (2.10) such that it maps into the space of all monomials *up to* degree d, by defining $k(x, x') = (\langle x, x' \rangle + 1)^d$ (Problem 2.17). Moreover, in practice, it is often useful to multiply the kernel by a scaling factor c to ensure that its numeric range is within some bounded interval, say $[-1, 1]$. The value of c will depend on the dimension and range of the data.

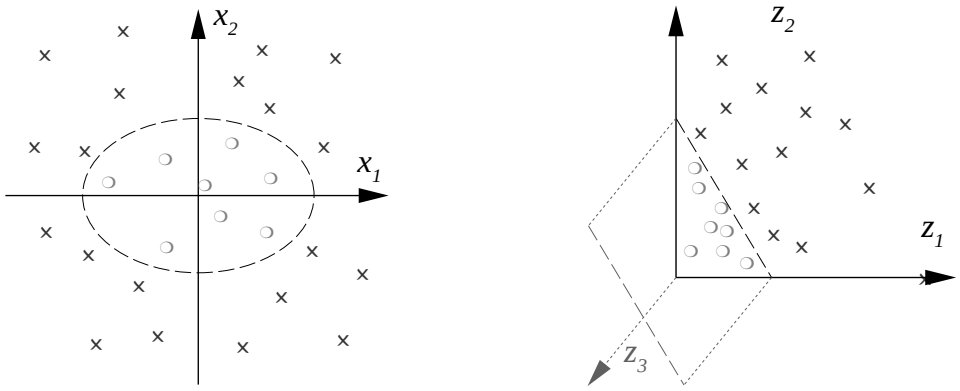


Figure 2.1 Toy example of a binary classification problem mapped into feature space. We assume that the true decision boundary is an ellipse in input space (left panel). The task of the learning process is to estimate this boundary based on empirical data consisting of training points in both classes (crosses and circles, respectively). When mapped into feature space via the nonlinear map $\Phi_2(x) = (z_1, z_2, z_3) = ([x]_1^2, [x]_2^2, \sqrt{2}\,[x]_1[x]_2)$ (right panel), the ellipse becomes a hyperplane (in the present simple case, it is parallel to the z_3 axis, hence all points are plotted in the (z_1, z_2) plane). This is due to the fact that ellipses can be written as linear equations in the entries of (z_1, z_2, z_3). Therefore, in feature space, the problem reduces to that of estimating a hyperplane from the mapped data points. Note that via the polynomial kernel (see (2.12) and (2.13)), the dot product in the three-dimensional space can be computed without computing Φ_2. Later in the book, we shall describe algorithms for constructing hyperplanes which are based on dot products (Chapter 7).

2.2 The Representation of Similarities in Linear Spaces

In what follows, we will look at things the other way round, and start with the kernel rather than with the feature map. Given some kernel, can we construct a feature space such that the kernel computes the dot product in that feature space; that is, such that (2.2) holds? This question has been brought to the attention of the machine learning community in a variety of contexts, especially during recent years [4, 152, 62, 561, 480]. In functional analysis, the same problem has been studied under the heading of *Hilbert space representations* of kernels. A good monograph on the theory of kernels is the book of Berg, Christensen, and Ressel [42]; indeed, a large part of the material in the present chapter is based on this work. We do not aim to be fully rigorous; instead, we try to provide insight into the basic ideas. As a rule, all the results that we state without proof can be found in [42]. Other standard references include [16, 455].

There is one more aspect in which this section differs from the previous one: the latter dealt with vectorial data, and the domain $\mathcal{X}$ was assumed to be a subset of $\mathbb{R}^N$. By contrast, the results in the current section hold for data drawn from domains which need no structure, other than their being nonempty sets. This generalizes kernel learning algorithms to a large number of situations where a vectorial representation is not readily available, and where one directly works

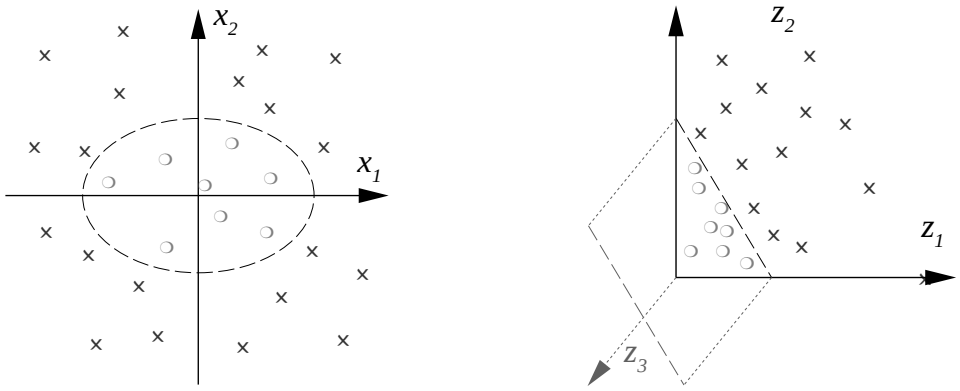


Figure 2.1 Toy example of a binary classification problem mapped into feature space. We assume that the true decision boundary is an ellipse in input space (left panel). The task of the learning process is to estimate this boundary based on empirical data consisting of training points in both classes (crosses and circles, respectively). When mapped into feature space via the nonlinear map $\Phi_2(x) = (z_1, z_2, z_3) = ([x]_1^2, [x]_2^2, \sqrt{2}\,[x]_1[x]_2)$ (right panel), the ellipse becomes a hyperplane (in the present simple case, it is parallel to the z_3 axis, hence all points are plotted in the (z_1, z_2) plane). This is due to the fact that ellipses can be written as linear equations in the entries of (z_1, z_2, z_3). Therefore, in feature space, the problem reduces to that of estimating a hyperplane from the mapped data points. Note that via the polynomial kernel (see (2.12) and (2.13)), the dot product in the three-dimensional space can be computed without computing Φ_2. Later in the book, we shall describe algorithms for constructing hyperplanes which are based on dot products (Chapter 7).

2.2 The Representation of Similarities in Linear Spaces

In what follows, we will look at things the other way round, and start with the kernel rather than with the feature map. Given some kernel, can we construct a feature space such that the kernel computes the dot product in that feature space; that is, such that (2.2) holds? This question has been brought to the attention of the machine learning community in a variety of contexts, especially during recent years [4, 152, 62, 561, 480]. In functional analysis, the same problem has been studied under the heading of *Hilbert space representations* of kernels. A good monograph on the theory of kernels is the book of Berg, Christensen, and Ressel [42]; indeed, a large part of the material in the present chapter is based on this work. We do not aim to be fully rigorous; instead, we try to provide insight into the basic ideas. As a rule, all the results that we state without proof can be found in [42]. Other standard references include [16, 455].

There is one more aspect in which this section differs from the previous one: the latter dealt with vectorial data, and the domain $\mathcal{X}$ was assumed to be a subset of $\mathbb{R}^N$. By contrast, the results in the current section hold for data drawn from domains which need no structure, other than their being nonempty sets. This generalizes kernel learning algorithms to a large number of situations where a vectorial representation is not readily available, and where one directly works

with pairwise distances or similarities between non-vectorial objects [246, 467, 154, 210, 234, 585]. This theme will recur in several places throughout the book, for instance in Chapter 13.

2.2.1 Positive Definite Kernels

We start with some basic definitions and results. As in the previous chapter, indices i and j are understood to run over $1, \ldots, m$.

Gram Matrix

Definition 2.3 (Gram Matrix) *Given a function $k : \mathcal{X}^2 \to \mathbb{K}$ (where $\mathbb{K} = \mathbb{C}$ or $\mathbb{K} = \mathbb{R}$) and patterns $x_1, \ldots, x_m \in \mathcal{X}$, the $m \times m$ matrix K with elements*

$$K_{ij} := k(x_i, x_j) \tag{2.14}$$

is called the Gram matrix *(or* kernel matrix*) of k with respect to $x_1, \ldots, x_m$.*

PD Matrix

Definition 2.4 (Positive Definite Matrix) *A complex $m \times m$ matrix K satisfying*

$$\sum_{i,j} c_i \bar{c}_j K_{ij} \geq 0 \tag{2.15}$$

for all $c_i \in \mathbb{C}$ is called positive definite.[1] *Similarly, a real symmetric $m \times m$ matrix K satisfying (2.15) for all $c_i \in \mathbb{R}$ is called* positive definite*.*

Note that a symmetric matrix is positive definite if and only if all its eigenvalues are nonnegative (Problem 2.4). The left hand side of (2.15) is often referred to as the *quadratic form* induced by K.

PD Kernel

Definition 2.5 ((Positive Definite) Kernel) *Let $\mathcal{X}$ be a nonempty set. A function k on $\mathcal{X} \times \mathcal{X}$ which for all $m \in \mathbb{N}$ and all $x_1 \ldots, x_m \in \mathcal{X}$ gives rise to a positive definite Gram matrix is called a* positive definite (pd) kernel*. Often, we shall refer to it simply as a* kernel*.*

Remark 2.6 (Terminology) *The term* kernel *stems from the first use of this type of function in the field of integral operators as studied by Hilbert and others [243, 359, 112]. A function k which gives rise to an operator T_k via*

$$(T_k f)(x) = \int_{\mathcal{X}} k(x, x') f(x')\, dx' \tag{2.16}$$

is called the kernel of T_k.

In the literature, a number of different terms are used for positive definite kernels, such as reproducing kernel*,* Mercer kernel*,* admissible kernel*,* Support Vector kernel*,* nonnegative definite kernel*, and* covariance function*. One might argue that the term* positive definite kernel *is slightly misleading. In matrix theory, the term* definite *is sometimes reserved for the case where equality in (2.15) only occurs if $c_1 = \ldots = c_m = 0$.*

1. The bar in $\bar{c}_j$ denotes complex conjugation; for real numbers, it has no effect.

Simply using the term positive kernel*, on the other hand, could be mistaken as referring to a kernel whose* values *are positive. Finally, the term positive* semi*definite kernel becomes rather cumbersome if it is to be used throughout a book. Therefore, we follow the convention used for instance in [42], and employ the term* positive definite *both for kernels and matrices in the way introduced above. The case where the value* 0 *is only attained if all coefficients are* 0 *will be referred to as* strictly *positive definite.*

We shall mostly use the term kernel*. Whenever we want to refer to a kernel $k(x, x')$ which is not positive definite in the sense stated above, it will be clear from the context.*

The definitions for positive definite kernels and positive definite matrices differ in the fact that in the former case, we are free to choose the points on which the kernel is evaluated — for every choice, the kernel induces a positive definite matrix.

Positive definiteness implies *positivity on the diagonal* (Problem 2.12),

$$k(x, x) \geq 0 \text{ for all } x \in \mathcal{X}, \tag{2.17}$$

and *symmetry* (Problem 2.13),

$$k(x_i, x_j) = \overline{k(x_j, x_i)}. \tag{2.18}$$

To also cover the complex-valued case, our definition of symmetry includes complex conjugation. The definition of symmetry of matrices is analogous; that is, $K_{ij} = \overline{K}_{ji}$.

Real-Valued Kernels

For real-valued kernels it is not sufficient to stipulate that (2.15) hold for real coefficients c_i. To get away with real coefficients only, we must additionally require that the kernel be symmetric (Problem 2.14); $k(x_i, x_j) = k(x_j, x_i)$ (cf. Problem 2.13).

It can be shown that whenever k is a (complex-valued) positive definite kernel, its real part is a (real-valued) positive definite kernel. Below, we shall largely be dealing with real-valued kernels. Most of the results, however, also apply for complex-valued kernels.

Kernels can be regarded as generalized dot products. Indeed, any dot product is a kernel (Problem 2.5); however, linearity in the arguments, which is a standard property of dot products, does not carry over to general kernels. However, another property of dot products, the Cauchy-Schwarz inequality, does have a natural generalization to kernels:

Proposition 2.7 (Cauchy-Schwarz Inequality for Kernels) *If k is a positive definite kernel, and $x_1, x_2 \in \mathcal{X}$, then*

$$|k(x_1, x_2)|^2 \leq k(x_1, x_1) \cdot k(x_2, x_2). \tag{2.19}$$

Proof For sake of brevity, we give a non-elementary proof using some basic facts of linear algebra. The 2×2 Gram matrix with entries $K_{ij} = k(x_i, x_j)$ $(i, j \in \{1, 2\})$ is positive definite. Hence both its eigenvalues are nonnegative, and so is their product, the determinant of K. Therefore

$$0 \leq K_{11}K_{22} - K_{12}K_{21} = K_{11}K_{22} - K_{12}\overline{K}_{12} = K_{11}K_{22} - |K_{12}|^2. \tag{2.20}$$

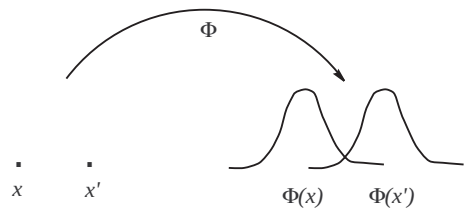


Figure 2.2 One instantiation of the feature map associated with a kernel is the map (2.21), which represents each pattern (in the picture, x or x') by a kernel-shaped *function* sitting on the pattern. In this sense, each pattern is represented by its similarity to *all* other patterns. In the picture, the kernel is assumed to be bell-shaped, e.g., a Gaussian $k(x, x') = \exp(-\|x - x'\|^2/(2\,\sigma^2))$. In the text, we describe the construction of a dot product $\langle .,. \rangle$ on the function space such that $k(x, x') = \langle \Phi(x), \Phi(x') \rangle$.

Substituting $k(x_i, x_j)$ for K_{ij}, we get the desired inequality. ■

We now show how the feature spaces in question are defined by the choice of kernel function.

2.2.2 The Reproducing Kernel Map

Feature Map

Assume that k is a real-valued positive definite kernel, and $\mathcal{X}$ a nonempty set. We define a map from $\mathcal{X}$ into the space of functions mapping $\mathcal{X}$ into $\mathbb{R}$, denoted as $\mathbb{R}^{\mathcal{X}} := \{f : \mathcal{X} \to \mathbb{R}\}$, via

$$\begin{aligned} \Phi : \mathcal{X} &\to \mathbb{R}^{\mathcal{X}} \\ x &\mapsto k(., x). \end{aligned} \tag{2.21}$$

Here, $\Phi(x)$ denotes the function that assigns the value $k(x', x)$ to $x' \in \mathcal{X}$, i.e., $\Phi(x)(.) = k(., x)$ (as shown in Figure 2.2).

We have thus turned each pattern into a function on the domain $\mathcal{X}$. In this sense, a pattern is now represented by its similarity to *all* other points in the input domain $\mathcal{X}$. This seems a very rich representation; nevertheless, it will turn out that the kernel allows the computation of the dot product in this representation. Below, we show how to construct a feature space associated with Φ, proceeding in the following steps:

1. Turn the image of Φ into a vector space,
2. define a dot product; that is, a strictly positive definite bilinear form, and
3. show that the dot product satisfies $k(x, x') = \langle \Phi(x), \Phi(x') \rangle$.

Vector Space

We begin by constructing a dot product space containing the images of the input patterns under Φ. To this end, we first need to define a vector space. This is done by taking linear combinations of the form

$$f(.) = \sum_{i=1}^{m} \alpha_i k(., x_i). \tag{2.22}$$

Here, $m \in \mathbb{N}$, $\alpha_i \in \mathbb{R}$ and $x_1, \ldots, x_m \in \mathcal{X}$ are arbitrary. Next, we define a dot product

between f and another function

$$g(.) = \sum_{j=1}^{m'} \beta_j k(., x'_j), \tag{2.23}$$

Dot Product

where $m' \in \mathbb{N}$, $\beta_j \in \mathbb{R}$ and $x'_1, \ldots, x'_{m'} \in \mathcal{X}$, as

$$\langle f, g \rangle := \sum_{i=1}^{m} \sum_{j=1}^{m'} \alpha_i \beta_j k(x_i, x'_j). \tag{2.24}$$

This expression explicitly contains the expansion coefficients, which need not be unique. To see that it is nevertheless well-defined, note that

$$\langle f, g \rangle = \sum_{j=1}^{m'} \beta_j f(x'_j), \tag{2.25}$$

using $k(x'_j, x_i) = k(x_i, x'_j)$. The sum in (2.25), however, does not depend on the particular expansion of f. Similarly, for g, note that

$$\langle f, g \rangle = \sum_{i=1}^{m} \alpha_i g(x_i). \tag{2.26}$$

The last two equations also show that $\langle \cdot, \cdot \rangle$ is bilinear. It is symmetric, as $\langle f, g \rangle = \langle g, f \rangle$. Moreover, it is positive definite, since positive definiteness of k implies that for any function f, written as (2.22), we have

$$\langle f, f \rangle = \sum_{i,j=1}^{m} \alpha_i \alpha_j k(x_i, x_j) \geq 0. \tag{2.27}$$

The latter implies that $\langle \cdot, \cdot \rangle$ is actually itself a positive definite kernel, defined on our space of functions. To see this, note that given functions $f_1, \ldots, f_n$, and coefficients $\gamma_1, \ldots, \gamma_n \in \mathbb{R}$, we have

$$\sum_{i,j=1}^{n} \gamma_i \gamma_j \langle f_i, f_j \rangle = \left\langle \sum_{i=1}^{n} \gamma_i f_i, \sum_{j=1}^{n} \gamma_j f_j \right\rangle \geq 0. \tag{2.28}$$

Here, the left hand equality follows from the bilinearity of $\langle \cdot, \cdot \rangle$, and the right hand inequality from (2.27). For the last step in proving that it qualifies as a dot product, we will use the following interesting property of Φ, which follows directly from the definition: for all functions (2.22), we have

$$\langle k(., x), f \rangle = f(x) \tag{2.29}$$

— k is the *representer of evaluation*. In particular,

$$\langle k(., x), k(., x') \rangle = k(x, x'). \tag{2.30}$$

Reproducing Kernel

By virtue of these properties, positive definite kernels k are also called *reproducing kernels* [16, 42, 455, 578, 467, 202]. By (2.29) and Proposition 2.7, we have

$$|f(x)|^2 = |\langle k(., x), f \rangle|^2 \leq k(x, x) \cdot \langle f, f \rangle . \tag{2.31}$$

Therefore, $\langle f, f \rangle = 0$ directly implies $f = 0$, which is the last property that required proof in order to establish that $\langle \cdot, \cdot \rangle$ is a dot product (cf. Section B.2).

The case of complex-valued kernels can be dealt with using the same construction; in that case, we will end up with a complex dot product space [42].

The above reasoning has shown that any positive definite kernel can be thought of as a dot product in another space: in view of (2.21), the reproducing kernel property (2.30) amounts to

$$\langle \Phi(x), \Phi(x') \rangle = k(x, x'). \tag{2.32}$$

Therefore, the dot product space $\mathcal{H}$ constructed in this way is one possible instantiation of the feature space associated with a kernel.

Kernels from Feature Maps

Above, we have started with the kernel, and constructed a feature map. Let us now consider the opposite direction. Whenever we have a mapping Φ from $\mathcal{X}$ into a dot product space, we obtain a positive definite kernel via $k(x, x') := \langle \Phi(x), \Phi(x') \rangle$. This can be seen by noting that for all $c_i \in \mathbb{R}, x_i \in \mathcal{X}, i = 1, \ldots, m$, we have

$$\sum_{i,j} c_i c_j k(x_i, x_j) = \left\langle \sum_i c_i \Phi(x_i), \sum_j c_j \Phi(x_j) \right\rangle = \left\| \sum_i c_i \Phi(x_i) \right\|^2 \geq 0, \tag{2.33}$$

due to the nonnegativity of the norm.

Equivalent Definition of PD Kernels

This has two consequences. First, it allows us to give an equivalent definition of positive definite kernels as functions with the property that there exists a map Φ into a dot product space such that (2.32) holds true. Second, it allows us to construct kernels from feature maps. For instance, it is in this way that powerful linear representations of 3D heads proposed in computer graphics [575, 59] give rise to kernels. The identity (2.32) forms the basis for the kernel trick:

Kernel Trick

Remark 2.8 ("Kernel Trick") *Given an algorithm which is formulated in terms of a positive definite kernel k, one can construct an alternative algorithm by replacing k by another positive definite kernel $\tilde{k}$.*

In view of the material in the present section, the justification for this procedure is the following: effectively, the original algorithm can be thought of as a dot product based algorithm operating on vectorial data $\Phi(x_1), \ldots, \Phi(x_m)$. The algorithm obtained by replacing k by $\tilde{k}$ then is exactly the same dot product based algorithm, only that it operates on $\tilde{\Phi}(x_1), \ldots, \tilde{\Phi}(x_m)$.

The best known application of the kernel trick is in the case where k is the dot product in the input domain (cf. Problem 2.5). The trick is not limited to that case, however: k and $\tilde{k}$ can *both* be nonlinear kernels. In general, care must be exercised in determining whether the resulting algorithm will be useful: sometimes, an algorithm will only work subject to additional conditions on the input data, e.g., the data set might have to lie in the positive orthant. We shall later see that certain kernels induce feature maps which enforce such properties for the mapped data (cf. (2.73)), and that there are algorithms which take advantage of these aspects (e.g., in Chapter 8). In such cases, not every conceivable positive definite kernel

will make sense.

Historical Remarks

Even though the kernel trick had been used in the literature for a fair amount of time [4, 62], it took until the mid 1990s before it was explicitly stated that *any* algorithm that only depends on dot products, i.e., any algorithm that is rotationally invariant, can be kernelized [479, 480]. Since then, a number of algorithms have benefitted from the kernel trick, such as the ones described in the present book, as well as methods for clustering in feature spaces [479, 215, 199].

Moreover, the machine learning community took time to comprehend that the definition of kernels on general sets (rather than dot product spaces) greatly extends the applicability of kernel methods [467], to data types such as texts and other sequences [234, 585, 23]. Indeed, this is now recognized as a crucial feature of kernels: they lead to an embedding of general data types in linear spaces.

Not surprisingly, the history of methods for representing kernels in linear spaces (in other words, the mathematical counterpart of the kernel trick) dates back significantly further than their use in machine learning. The methods appear to have first been studied in the 1940s by Kolmogorov [304] for countable $\mathcal{X}$ and Aronszajn [16] in the general case. Pioneering work on linear representations of a related class of kernels, to be described in Section 2.4, was done by Schoenberg [465]. Further bibliographical comments can be found in [42].

We thus see that the mathematical basis for kernel algorithms has been around for a long time. As is often the case, however, the practical importance of mathematical results was initially underestimated.[2]

2.2.3 Reproducing Kernel Hilbert Spaces

In the last section, we described how to define a space of functions which is a valid realization of the feature spaces associated with a given kernel. To do this, we had to make sure that the space is a vector space, and that it is endowed with a dot product. Such spaces are referred to as dot product spaces (cf. Appendix B), or equivalently as *pre-Hilbert* spaces. The reason for the latter is that one can turn them into Hilbert spaces (cf. Section B.3) by a fairly simple mathematical trick. This additional structure has some mathematical advantages. For instance, in Hilbert spaces it is always possible to define projections. Indeed, Hilbert spaces are one of the favorite concepts of functional analysis.

So let us again consider the pre-Hilbert space of functions (2.22), endowed with the dot product (2.24). To turn it into a Hilbert space (over $\mathbb{R}$), one *completes* it in the norm corresponding to the dot product, $\|f\| := \sqrt{\langle f, f\rangle}$. This is done by adding the limit points of sequences that are convergent in that norm (see Appendix B).

2. This is illustrated by the following quotation from an excellent machine learning textbook published in the seventies (p. 174 in [152]): *"The familiar functions of mathematical physics are eigenfunctions of symmetric kernels, and their use is often suggested for the construction of potential functions. However, these suggestions are more appealing for their mathematical beauty than their practical usefulness."*

RKHS

In view of the properties (2.29) and (2.30), this space is usually called a *reproducing kernel Hilbert space (RKHS)*.

In general, an RKHS can be defined as follows.

Definition 2.9 (Reproducing Kernel Hilbert Space) *Let $\mathcal{X}$ be a nonempty set (often called the index set) and by $\mathcal{H}$ a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$. Then $\mathcal{H}$ is called a reproducing kernel Hilbert space endowed with the dot product $\langle \cdot, \cdot \rangle$ (and the norm $\|f\| := \sqrt{\langle f, f \rangle}$) if there exists a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with the following properties.*

Reproducing Property

1. *k has the reproducing property*[3]

$$\langle f, k(x, \cdot) \rangle = f(x) \text{ for all } f \in \mathcal{H}; \tag{2.34}$$

in particular,

$$\langle k(x, \cdot), k(x', \cdot) \rangle = k(x, x'). \tag{2.35}$$

Closed Space

2. *k spans $\mathcal{H}$, i.e. $\mathcal{H} = \overline{\text{span}\{k(x, \cdot) | x \in \mathcal{X}\}}$ where $\overline{X}$ denotes the completion of the set X (cf. Appendix B).*

On a more abstract level, an RKHS can be defined as a Hilbert space of functions f on $\mathcal{X}$ such that all evaluation functionals (the maps $f \mapsto f(x')$, where $x' \in \mathcal{X}$) are continuous. In that case, by the Riesz representation theorem (e.g., [429]), for each $x' \in \mathcal{X}$ there exists a unique function of x, called $k(x, x')$, such that

$$f(x') = \langle f, k(., x') \rangle . \tag{2.36}$$

It follows directly from (2.35) that $k(x, x')$ is symmetric in its arguments (see Problem 2.28) and satisfies the conditions for positive definiteness.

Uniqueness of k

Note that the RKHS uniquely determines k. This can be shown by contradiction: assume that there exist two kernels, say k and k', spanning the same RKHS $\mathcal{H}$. From Problem 2.28 we know that both k and k' must be symmetric. Moreover, from (2.34) we conclude that

$$\langle k(x, \cdot), k'(x', \cdot) \rangle_{\mathcal{H}} = k(x, x') = k'(x', x). \tag{2.37}$$

In the second equality we used the symmetry of the dot product. Finally, symmetry in the arguments of k yields $k(x, x') = k'(x, x')$ which proves our claim.

2.2.4 The Mercer Kernel Map

Section 2.2.2 has shown that any positive definite kernel can be represented as a dot product in a linear space. This was done by explicitly constructing a (Hilbert) space that does the job. The present section will construct another Hilbert space.

3. Note that this implies that each $f \in \mathcal{H}$ is actually a single function whose values at any $x \in \mathcal{X}$ are well-defined. In contrast, L_2 Hilbert spaces usually do not have this property. The elements of these spaces are equivalence classes of functions that disagree only on sets of measure 0; cf. footnote 15 in Section B.3.

One could argue that this is superfluous, given that any two separable Hilbert spaces are isometrically isomorphic, in other words, it is possible to define a one-to-one linear map between the spaces which preserves the dot product. However, the tool that we shall presently use, Mercer's theorem, has played a crucial role in the understanding of SVMs, and it provides valuable insight into the geometry of feature spaces, which more than justifies its detailed discussion. In the SVM literature, the kernel trick is usually introduced via Mercer's theorem.

Mercer's Theorem

We start by stating the version of Mercer's theorem given in [606]. We assume $(\mathcal{X}, \mu)$ to be a finite measure space.[4] The term *almost all* (cf. Appendix B) means *except for sets of measure zero*. For the commonly used Lebesgue-Borel measure, countable sets of individual points are examples of zero measure sets. Note that the integral with respect to a measure is explained in Appendix B. Readers who do not want to go into mathematical detail may simply want to think of the $d\mu(x')$ as a dx', and of $\mathcal{X}$ as a compact subset of $\mathbb{R}^N$. For further explanations of the terms involved in this theorem, cf. Appendix B, especially Section B.3.

Theorem 2.10 (Mercer [359, 307]) *Suppose $k \in L_\infty(\mathcal{X}^2)$ is a symmetric real-valued function such that the integral operator (cf. (2.16))*

$$\begin{aligned} T_k &: L_2(\mathcal{X}) \to L_2(\mathcal{X}) \\ (T_k f)(x) &:= \int_{\mathcal{X}} k(x, x') f(x')\, d\mu(x') \end{aligned} \tag{2.38}$$

is positive definite*; that is, for all $f \in L_2(\mathcal{X})$, we have*

$$\int_{\mathcal{X}^2} k(x, x') f(x) f(x')\, d\mu(x) d\mu(x') \geq 0. \tag{2.39}$$

Let $\psi_j \in L_2(\mathcal{X})$ be the normalized orthogonal eigenfunctions of T_k associated with the eigenvalues $\lambda_j > 0$, sorted in non-increasing order. Then

1. $(\lambda_j)_j \in \ell_1$,
2. $k(x, x') = \sum_{j=1}^{N_{\mathcal{H}}} \lambda_j \psi_j(x) \psi_j(x')$ *holds for almost all (x, x'). Either $N_{\mathcal{H}} \in \mathbb{N}$, or $N_{\mathcal{H}} = \infty$; in the latter case, the series converges absolutely and uniformly for almost all (x, x').*

For the converse of Theorem 2.10, see Problem 2.23. For a data-dependent approximation and its relationship to kernel PCA (Section 1.7), see Problem 2.26.

From statement 2 it follows that $k(x, x')$ corresponds to a dot product in $\ell_2^{N_{\mathcal{H}}}$, since $k(x, x') = \langle \Phi(x), \Phi(x') \rangle$ with

$$\begin{aligned} \Phi : \mathcal{X} &\to \ell_2^{N_{\mathcal{H}}} \\ x &\mapsto (\sqrt{\lambda_j} \psi_j(x))_{j=1,\ldots,N_{\mathcal{H}}}, \end{aligned} \tag{2.40}$$

for almost all $x \in \mathcal{X}$. Note that we use the same Φ as in (2.21) to denote the feature

4. A finite measure space is a set $\mathcal{X}$ with a σ-algebra (Definition B.1) defined on it, and a measure (Definition B.2) defined on the latter, satisfying $\mu(\mathcal{X}) < \infty$ (so that, up to a scaling factor, μ is a probability measure).

map, although the target spaces are different. However, this distinction is not important for the present purposes — we are interested in the existence of some Hilbert space in which the kernel corresponds to the dot product, and not in what particular representation of it we are using.

In fact, it has been noted [467] that the *uniform* convergence of the series implies that given any $\epsilon > 0$, there exists an $n \in \mathbb{N}$ such that even if $N_{\mathcal{H}} = \infty$, k can be approximated within accuracy ϵ as a dot product in $\mathbb{R}^n$: for almost all $x, x' \in \mathcal{X}$, $|k(x,x') - \langle \Phi^n(x), \Phi^n(x') \rangle| < \epsilon$, where $\Phi^n : x \mapsto (\sqrt{\lambda_1}\psi_1(x), \ldots, \sqrt{\lambda_n}\psi_n(x))$. The feature space can thus always be thought of as finite-dimensional within some accuracy ϵ. We summarize our findings in the following proposition.

Mercer Feature Map

Proposition 2.11 (Mercer Kernel Map) *If k is a kernel satisfying the conditions of Theorem 2.10, we can construct a mapping Φ into a space where k acts as a dot product,*

$$\langle \Phi(x), \Phi(x') \rangle = k(x,x'), \tag{2.41}$$

for almost all $x, x' \in \mathcal{X}$. Moreover, given any $\epsilon > 0$, there exists a map Φ_n into an n-dimensional dot product space (where $n \in \mathbb{N}$ depends on ϵ) such that

$$|k(x,x') - \langle \Phi^n(x), \Phi^n(x') \rangle| < \epsilon \tag{2.42}$$

for almost all $x, x' \in \mathcal{X}$.

Both Mercer kernels and positive definite kernels can thus be represented as dot products in Hilbert spaces. The following proposition, showing a case where the two types of kernels coincide, thus comes as no surprise.

Proposition 2.12 (Mercer Kernels are Positive Definite [359, 42]) *Let $\mathcal{X} = [a,b]$ be a compact interval and let $k : [a,b] \times [a,b] \to \mathbb{C}$ be continuous. Then k is a positive definite kernel if and only if*

$$\int_a^b \int_a^b k(x,x') f(x) f(x') \, dx \, dx' \geq 0 \tag{2.43}$$

for each continuous function $f : \mathcal{X} \to \mathbb{C}$.

Note that the conditions in this proposition are actually more restrictive than those of Theorem 2.10. Using the feature space representation (Proposition 2.11), however, it is easy to see that Mercer kernels are also positive definite (for almost all $x, x' \in \mathcal{X}$) in the more general case of Theorem 2.10: given any $\mathbf{c} \in \mathbb{R}^m$, we have

$$\sum_{i,j} c_i c_j k(x_i, x_j) = \sum_{i,j} c_i c_j \langle \Phi(x_i), \Phi(x_j) \rangle = \left\| \sum_i c_i \Phi(x_i) \right\|^2 \geq 0. \tag{2.44}$$

Being positive definite, Mercer kernels are thus also reproducing kernels.

We next show how the reproducing kernel map is related to the Mercer kernel map constructed from the eigenfunction decomposition [202, 467]. To this end, let us consider a kernel which satisfies the condition of Theorem 2.10, and construct

a dot product $\langle \cdot, \cdot \rangle$ such that k becomes a reproducing kernel for the Hilbert space $\mathcal{H}$ containing the functions

$$f(x) = \sum_{i=1}^{\infty} \alpha_i k(x, x_i) = \sum_{i=1}^{\infty} \alpha_i \sum_{j=1}^{N_{\mathcal{H}}} \lambda_j \psi_j(x) \psi_j(x_i). \tag{2.45}$$

By linearity, which holds for any dot product, we have

$$\langle f, k(., x') \rangle = \sum_{i=1}^{\infty} \alpha_i \sum_{j,n=1}^{N_{\mathcal{H}}} \lambda_j \psi_j(x_i) \langle \psi_j, \psi_n \rangle \lambda_n \psi_n(x'). \tag{2.46}$$

Since k is a Mercer kernel, the ψ_i ($i = 1, \ldots, N_{\mathcal{H}}$) can be chosen to be orthogonal with respect to the dot product in $L_2(\mathcal{X})$. Hence it is straightforward to choose $\langle \cdot, \cdot \rangle$ such that

$$\langle \psi_j, \psi_n \rangle = \delta_{jn} / \lambda_j \tag{2.47}$$

(using the Kronecker symbol δ_{jn}, see (B.30)), in which case (2.46) reduces to the reproducing kernel property (2.36) (using (2.45)). For a coordinate representation in the RKHS, see Problem 2.29.

Equivalence of Feature Spaces

The above connection between the Mercer kernel map and the RKHS map is instructive, but we shall rarely make use of it. In fact, we will usually *identify* the different feature spaces. Thus, to avoid confusion in subsequent chapters, the following comments are necessary. As described above, there are different ways of constructing feature spaces for any given kernel. In fact, they can even differ in terms of their dimensionality (cf. Problem 2.22). The two feature spaces that we will mostly use in this book are the RKHS associated with k (Section 2.2.2) and the Mercer ℓ_2 feature space. We will mostly use the same symbol $\mathcal{H}$ for all feature spaces that are associated with a given kernel. This makes sense provided that everything we do, at the end of the day, reduces to dot products. For instance, let us assume that Φ_1, Φ_2 are maps into the feature spaces $\mathcal{H}_1, \mathcal{H}_2$ respectively, both associated with the kernel k; in other words,

$$k(x, x') = \langle \Phi_i(x), \Phi_i(x') \rangle_{\mathcal{H}_i}, \text{ for } i = 1, 2. \tag{2.48}$$

Then it will usually *not* be the case that $\Phi_1(x) = \Phi_2(x)$; due to (2.48), however, we always have $\langle \Phi_1(x), \Phi_1(x') \rangle_{\mathcal{H}_1} = \langle \Phi_2(x), \Phi_2(x') \rangle_{\mathcal{H}_2}$. Therefore, as long as we are only interested in dot products, the two spaces can be considered identical.

An example of this identity is the so-called large margin regularizer that is usually used in SVMs, as discussed in the introductory chapter (cf. also Chapters 4 and 7),

$$\langle \mathbf{w}, \mathbf{w} \rangle, \text{ where } \mathbf{w} = \sum_{i=1}^{m} \alpha_i \Phi(x_i). \tag{2.49}$$

No matter whether Φ is the RKHS map $\Phi(x_i) = k(., x_i)$ (2.21) or the Mercer map $\Phi(x_i) = (\sqrt{\lambda_j} \psi_j(x))_{j=1,\ldots,N_{\mathcal{H}}}$ (2.40), the value of $\|\mathbf{w}\|^2$ will not change.

This point is of great importance, and we hope that all readers are still with us.

It is fair to say, however, that Section 2.2.5 can be skipped at first reading.

2.2.5 The Shape of the Mapped Data in Feature Space

Using Mercer's theorem, we have shown that one can think of the feature map as a map into a high- or infinite-dimensional Hilbert space. The argument in the remainder of the section shows that this typically entails that the mapped data $\Phi(\mathcal{X})$ lie in some box with rapidly decaying side lengths [606]. By this we mean that the range of the data decreases as the dimension index j increases, with a rate that depends on the size of the eigenvalues.

Let us assume that for all $j \in \mathbb{N}$, we have $\sup_{x\in\mathcal{X}} \lambda_j |\psi_j(x)|^2 < \infty$. Define the sequence

$$l_j := \sup_{x\in\mathcal{X}} \lambda_j |\psi_j(x)|^2. \tag{2.50}$$

Note that if

$$C_k := \sup_j \sup_{x\in\mathcal{X}} |\psi_j(x)| \tag{2.51}$$

exists (see Problem 2.24), then we have $l_j \le \lambda_j C_k^2$. However, if the λ_j decay rapidly, then (2.50) can be finite even if (2.51) is not.

By construction, $\Phi(\mathcal{X})$ is contained in an axis parallel parallelepiped in $\ell_2^{N_{\mathfrak{H}}}$ with side lengths $2\sqrt{l_j}$ (cf. (2.40)).[5]

Consider an example of a common kernel, the Gaussian, and let μ (see Theorem 2.10) be the Lebesgue measure. In this case, the eigenvectors are sine and cosine functions (with supremum one), and thus the sequence of the l_j coincides with the sequence of the eigenvalues λ_j. Generally, whenever $\sup_{x\in\mathcal{X}} |\psi_j(x)|^2$ is finite, the l_j decay as fast as the λ_j. We shall see in Sections 4.4, 4.5 and Chapter 12 that for many common kernels, this decay is very rapid.

It will be useful to consider operators that map $\Phi(\mathcal{X})$ into balls of some radius R centered at the origin. The following proposition characterizes a class of such operators, determined by the sequence $(l_j)_{j\in\mathbb{N}}$. Recall that $\mathbb{R}^{\mathbb{N}}$ denotes the space of all real sequences.

Proposition 2.13 (Mapping $\Phi(\mathcal{X})$ into ℓ_2) *Let S be the diagonal map*

$$\begin{array}{rccc} S: & \mathbb{R}^{\mathbb{N}} & \to & \mathbb{R}^{\mathbb{N}} \\ & (x_j)_j & \mapsto & S(x_j)_j = (s_j x_j)_j, \end{array} \tag{2.52}$$

where $(s_j)_j \in \mathbb{R}^{\mathbb{N}}$. If $\left(s_j\sqrt{l_j}\right)_j \in \ell_2$, then S maps $\Phi(\mathcal{X})$ into a ball centered at the origin whose radius is $R = \left\| \left(s_j\sqrt{l_j}\right)_j \right\|$.

5. In fact, it is sufficient to use the essential supremum in (2.50). In that case, subsequent statements also only hold true almost everywhere.

Proof Suppose $\left(s_j\sqrt{l_j}\right)_j \in \ell_2$. Using the Mercer map (2.40), we have

$$\|S\Phi(x)\|^2 = \sum_{j\in\mathbb{N}} s_j^2\lambda_j|\psi_j(x)|^2 \leq \sum_{j\in\mathbb{N}} s_j^2 l_j = R \tag{2.53}$$

for any $x \in \mathcal{X}$. Hence $S\Phi(\mathcal{X}) \subseteq \ell_2$. ■

The converse is not necessarily the case. To see this, note that if $\left(s_j\sqrt{l_j}\right)_j \notin \ell_2$, amounting to saying that

$$\sum_j s_j^2 \sup_{x\in\mathcal{X}} \lambda_j|\psi_j(x)|^2 \tag{2.54}$$

is not finite, then there need not always exist an $x \in \mathcal{X}$ such that $S\Phi(x) = \left(s_j\sqrt{\lambda_j}\psi_j(x)\right)_j \notin \ell_2$, i.e., that

$$\sum_j s_j^2\lambda_j|\psi_j(x)|^2 \tag{2.55}$$

is not finite.

To see how the freedom to rescale $\Phi(\mathcal{X})$ effectively restricts the class of functions we are using, we first note that everything in the feature space $\mathcal{H} = \ell_2^{N_\mathcal{H}}$ is done in terms of dot products. Therefore, we can compensate any invertible symmetric linear transformation of the data in $\mathcal{H}$ by the inverse transformation on the set of admissible weight vectors in $\mathcal{H}$. In other words, for any invertible symmetric operator S on $\mathcal{H}$, we have $\left\langle S^{-1}\mathbf{w}, S\Phi(x)\right\rangle = \langle \mathbf{w}, \Phi(x)\rangle$ for all $x \in \mathcal{X}$.

As we shall see below (cf. Theorem 5.5, Section 12.4, and Problem 7.5), there exists a class of generalization error bound that depends on the radius R of the smallest sphere containing the data. If the $(l_i)_i$ decay rapidly, we are not actually "making use" of the whole sphere. In this case, we may construct a diagonal scaling operator S which inflates the sides of the above parallelepiped as much as possible, while ensuring that it is still contained within a sphere of the original radius R in $\mathcal{H}$ (Figure 2.3). By effectively reducing the size of the function class, this will provide a way of strengthening the bounds. A similar idea, using kernel PCA (Section 14.2) to determine empirical scaling coefficients, has been successfully applied by [101].

We conclude this section with another useful insight that characterizes a property of the feature map Φ. Note that most of what was said so far applies to the case where the input domain $\mathcal{X}$ is a general set. In this case, it is not possible to make nontrivial statements about continuity properties of Φ. This changes if we assume $\mathcal{X}$ to be endowed with a notion of closeness, by turning it into a so-called topological space. Readers not familiar with this concept will be reassured to hear that Euclidean vector spaces are particular cases of topological spaces.

Continuity of Φ

Proposition 2.14 (Continuity of the Feature Map [402]) *If $\mathcal{X}$ is a topological space and k is a continuous positive definite kernel on $\mathcal{X}\times\mathcal{X}$, then there exists a Hilbert space $\mathcal{H}$ and a continuous map $\Phi : \mathcal{X} \to \mathcal{H}$ such that for all $x, x' \in \mathcal{X}$, we have $k(x, x') = \langle\Phi(x), \Phi(x')\rangle$.*

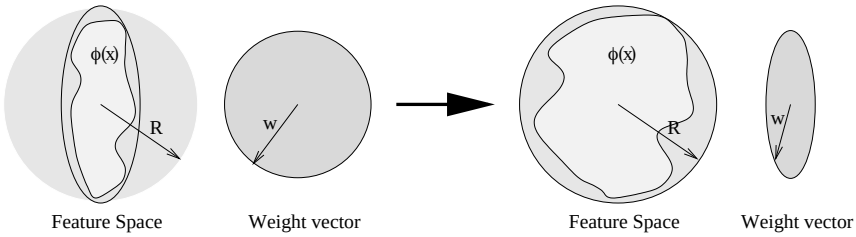


Figure 2.3 Since everything is done in terms of dot products, scaling up the data by an operator S can be compensated by scaling the weight vectors with S^{-1} (cf. text). By choosing S such that the data are still contained in a ball of the same radius R, we effectively reduce our function class (parametrized by the weight vector), which can lead to better generalization bounds, depending on the kernel inducing the map Φ.

2.2.6 The Empirical Kernel Map

The map Φ, defined in (2.21), transforms each input pattern into a function on $\mathcal{X}$, that is, into a potentially infinite-dimensional object. For any given set of points, however, it is possible to approximate Φ by only evaluating it on these points (cf. [232, 350, 361, 547, 474]):

Empirical Kernel Map

Definition 2.15 (Empirical Kernel Map) *For a given set* $\{z_1, \ldots, z_n\} \subset \mathcal{X}$, $n \in \mathbb{N}$, *we call*

$$\Phi_n : \mathbb{R}^N \to \mathbb{R}^n \text{ where } x \mapsto k(., x)|_{\{z_1,\ldots,z_n\}} = (k(z_1, x), \ldots, k(z_n, x))^\top \tag{2.56}$$

the empirical kernel map *w.r.t.* $\{z_1, \ldots, z_n\}$.

As an example, consider first the case where k is a positive definite kernel, and $\{z_1, \ldots, z_n\} = \{x_1, \ldots, x_m\}$; we thus evaluate $k(., x)$ on the training patterns. If we carry out a linear algorithm in feature space, then everything will take place in the linear span of the mapped training patterns. Therefore, we can represent the $k(., x)$ of (2.21) as $\Phi_m(x)$ without losing information. The dot product to use in that representation, however, is not simply the canonical dot product in $\mathbb{R}^m$, since the $\Phi(x_i)$ will usually not form an orthonormal system. To turn Φ_m into a feature map associated with k, we need to endow $\mathbb{R}^m$ with a dot product $\langle \cdot, \cdot \rangle_m$ such that

$$k(x, x') = \langle \Phi_m(x), \Phi_m(x') \rangle_m . \tag{2.57}$$

To this end, we use the ansatz $\langle \cdot, \cdot \rangle_m = \langle \cdot, M \cdot \rangle$, with M being a positive definite matrix.[6] Enforcing (2.57) on the training patterns, this yields the self-consistency condition [478, 512]

$$K = KMK, \tag{2.58}$$

6. Every dot product in $\mathbb{R}^m$ can be written in this form. We do not require strict definiteness of M, as the null space can be projected out, leading to a lower-dimensional feature space.

where K is the Gram matrix. The condition (2.58) can be satisfied for instance by the (pseudo-)inverse $M = K^{-1}$. Equivalently, we could have incorporated this rescaling operation, which corresponds to a Kernel PCA "whitening" ([478, 547, 474], cf. Section 11.4), directly into the map, by whitening (2.56) to get

Kernel PCA Map

$$\Phi_m^w : x \mapsto K^{-\frac{1}{2}}(k(x_1, x), \ldots, k(x_m, x)). \tag{2.59}$$

This simply amounts to dividing the eigenvector basis vectors of K by $\sqrt{\lambda_i}$, where the λ_i are the eigenvalues of K.[7] This parallels the rescaling of the eigenfunctions of the integral operator belonging to the kernel, given by (2.47). It turns out that this map can equivalently be performed using kernel PCA feature extraction (see Problem 14.8), which is why we refer to this map as the *kernel PCA map*.

Note that we have thus constructed a data-dependent feature map into an m-dimensional space which satisfies $\langle \Phi_m^w(x), \Phi_m^w(x') \rangle = k(x, x')$, i.e., we have found an m-dimensional feature space associated with the given kernel. In the case where K is invertible, $\Phi_m^w(x)$ computes the coordinates of $\Phi(x)$ when represented in a basis of the m-dimensional subspace spanned by $\Phi(x_1), \ldots, \Phi(x_m)$.

For data sets where the number of examples is smaller than their dimension, it can actually be computationally attractive to carry out Φ_m^w explicitly, rather than using kernels in subsequent algorithms. Moreover, algorithms which are not readily "kernelized" may benefit from explicitly carrying out the kernel PCA map.

We end this section with two notes which illustrate why the use of (2.56) need not be restricted to the special case we just discussed.

- *More general kernels.* When using non-symmetric kernels k in (2.56), together with the canonical dot product, we effectively work with the positive definite matrix $K^\top K$. Note that each positive definite matrix can be written as $K^\top K$. Therefore, working with positive definite kernels leads to an equally rich set of nonlinearities as working with an empirical kernel map using general non-symmetric kernels. If we wanted to carry out the whitening step, we would have to use $(K^\top K)^{-1/4}$ (cf. footnote 7 concerning potential singularities).
- *Different evaluation sets.* Things can be sped up by using expansion sets of the form $\{z_1, \ldots, z_n\}$, mapping into an n-dimensional space, with $n < m$, as done in [100, 228]. In that case, one modifies (2.59) to

$$\Phi_n^w : x \mapsto K_n^{-\frac{1}{2}}(k(z_1, x), \ldots, k(z_n, x)), \tag{2.60}$$

where $(K_n)_{ij} := k(z_i, z_j)$. The expansion set can either be a subset of the training set,[8] or some other set of points. We will later return to the issue of how to choose

7. It is understood that if K is singular, we use the pseudo-inverse of $K^{1/2}$ in which case we get an even lower dimensional subspace.
8. In [228] it is recommended that the size n of the expansion set is chosen large enough to ensure that the smallest eigenvalue of K_n is larger than some predetermined $\epsilon > 0$. Alternatively, one can start off with a larger set, and use kernel PCA to *select* the most important components for the map, see Problem 14.8. In the kernel PCA case, the map (2.60) is com-

the best set (see Section 10.2 and Chapter 18). As an aside, note that in the case of Kernel PCA (see Section 1.7 and Chapter 14 below), one does not need to worry about the whitening step in (2.59) and (2.60): using the canonical dot product in $\mathbb{R}^m$ (rather than $\langle \cdot, \cdot \rangle$) will simply lead to diagonalizing K^2 instead of K, which yields the same eigenvectors with squared eigenvalues. This was pointed out by [350, 361]. The study [361] reports experiments where (2.56) was employed to speed up Kernel PCA by choosing $\{z_1, \ldots, z_n\}$ as a subset of $\{x_1, \ldots, x_m\}$.

2.2.7 A Kernel Map Defined from Pairwise Similarities

In practice, we are given a finite amount of data $x_1, \ldots, x_m$. The following simple observation shows that even if we do not want to (or are unable to) analyze a given kernel k analytically, we can still compute a map Φ such that k corresponds to a dot product in the linear span of the $\Phi(x_i)$:

Proposition 2.16 (Data-Dependent Kernel Map [467]) *Suppose the data $x_1, \ldots, x_m$ and the kernel k are such that the kernel Gram matrix $K_{ij} = k(x_i, x_j)$ is positive definite. Then it is possible to construct a map Φ into an m-dimensional feature space $\mathcal{H}$ such that*

$$k(x_i, x_j) = \left\langle \Phi(x_i), \Phi(x_j) \right\rangle. \tag{2.61}$$

Conversely, given an arbitrary map Φ into some feature space $\mathcal{H}$, the matrix $K_{ij} = \left\langle \Phi(x_i), \Phi(x_j) \right\rangle$ is positive definite.

Proof First assume that K is positive definite. In this case, it can be diagonalized as $K = SDS^\top$, with an orthogonal matrix S and a diagonal matrix D with nonnegative entries. Then

$$k(x_i, x_j) = (SDS^\top)_{ij} = \langle S_i, DS_j \rangle = \left\langle \sqrt{D} S_i, \sqrt{D} S_j \right\rangle, \tag{2.62}$$

where we have defined the S_i as the rows of S (note that the columns of S would be K's eigenvectors). Therefore, K is the Gram matrix of the vectors $\sqrt{D_{ii}} \cdot S_i$.[9] Hence the following map Φ, defined on $x_1, \ldots, x_m$ will satisfy (2.61)

$$\Phi : x_i \mapsto \sqrt{D_{ii}} \cdot S_i. \tag{2.63}$$

Thus far, Φ is only defined on a set of points, rather than on a vector space. Therefore, it makes no sense to ask whether it is linear. We can, however, ask whether it can be *extended* to a linear map, provided the x_i are elements of a vector space. The answer is that if the x_i are linearly dependent (which is often the case), then this will not be possible, since a linear map would then typically be over-

puted as $D_n^{-1/2} U_n^\top (k(z_1, x), \ldots, k(z_n, x))$, where $U_n D_n U_n^\top$ is the eigenvalue decomposition of K_n. Note that the columns of U_n are the eigenvectors of K_n. We discard all columns that correspond to zero eigenvalues, as well as the corresponding dimensions of D_n. To *approximate* the map, we may actually discard all eigenvalues smaller than some $\epsilon > 0$.

9. In fact, every positive definite matrix is the Gram matrix of some set of vectors [46].

determined by the m conditions (2.63).

For the converse, assume an arbitrary $\boldsymbol{\alpha} \in \mathbb{R}^m$, and compute

$$\sum_{i,j=1}^{m} \alpha_i \alpha_j K_{ij} = \left\langle \sum_{i=1}^{m} \alpha_i \Phi(x_i), \sum_{j=1}^{m} \alpha_j \Phi(x_j) \right\rangle = \left\| \sum_{i=1}^{m} \alpha_i \Phi(x_i) \right\|^2 \geq 0. \tag{2.64}$$

■

In particular, this result implies that given data $x_1, \ldots, x_m$, and a kernel k which gives rise to a positive definite matrix K, it is always possible to construct a feature space $\mathcal{H}$ of dimension at most m that we are implicitly working in when using kernels (cf. Problem 2.32 and Section 2.2.6).

If we perform an algorithm which requires k to correspond to a dot product in some other space (as for instance the SV algorithms described in this book), it is possible that even though k is not positive definite in general, it still gives rise to a positive definite Gram matrix K with respect to the training data at hand. In this case, Proposition 2.16 tells us that nothing will go wrong during training when we work with these data. Moreover, if k leads to a matrix with some small negative eigenvalues, we can add a small multiple of some strictly positive definite kernel k' (such as the identity $k'(x_i, x_j) = \delta_{ij}$) to obtain a positive definite matrix. To see this, suppose that $\lambda_{\min} < 0$ is the minimal eigenvalue of k's Gram matrix. Note that being strictly positive definite, the Gram matrix K' of k' satisfies

$$\min_{\|\boldsymbol{\alpha}\|=1} \langle \boldsymbol{\alpha}, K'\boldsymbol{\alpha} \rangle \geq \lambda'_{\min} > 0, \tag{2.65}$$

where $\lambda'_{\min}$ denotes its minimal eigenvalue, and the first inequality follows from Rayleigh's principle (B.57). Therefore, provided that $\lambda_{\min} + \lambda\lambda'_{\min} \geq 0$, we have

$$\langle \boldsymbol{\alpha}, (K + \lambda K')\boldsymbol{\alpha} \rangle = \langle \boldsymbol{\alpha}, K\boldsymbol{\alpha} \rangle + \lambda \langle \boldsymbol{\alpha}, K'\boldsymbol{\alpha} \rangle \geq \|\boldsymbol{\alpha}\|^2 \left(\lambda_{\min} + \lambda\lambda'_{\min}\right) \geq 0 \tag{2.66}$$

for all $\boldsymbol{\alpha} \in \mathbb{R}^m$, rendering $(K + \lambda K')$ positive definite.

2.3 Examples and Properties of Kernels

Polynomial

For the following examples, let us assume that $\mathcal{X} \subset \mathbb{R}^N$. Besides homogeneous polynomial kernels (cf. Proposition 2.1),

$$k(x, x') = \langle x, x' \rangle^d, \tag{2.67}$$

Gaussian

Boser, Guyon, and Vapnik [62, 223, 561] suggest the usage of Gaussian radial basis function kernels [26, 4],

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\,\sigma^2}\right), \tag{2.68}$$

Sigmoid

where $\sigma > 0$, and sigmoid kernels,

$$k(x, x') = \tanh(\kappa \langle x, x' \rangle + \vartheta), \tag{2.69}$$

determined by the m conditions (2.63).

For the converse, assume an arbitrary $\boldsymbol{\alpha} \in \mathbb{R}^m$, and compute

$$\sum_{i,j=1}^{m} \alpha_i \alpha_j K_{ij} = \left\langle \sum_{i=1}^{m} \alpha_i \Phi(x_i), \sum_{j=1}^{m} \alpha_j \Phi(x_j) \right\rangle = \left\| \sum_{i=1}^{m} \alpha_i \Phi(x_i) \right\|^2 \geq 0. \tag{2.64}$$

■

In particular, this result implies that given data $x_1, \ldots, x_m$, and a kernel k which gives rise to a positive definite matrix K, it is always possible to construct a feature space $\mathcal{H}$ of dimension at most m that we are implicitly working in when using kernels (cf. Problem 2.32 and Section 2.2.6).

If we perform an algorithm which requires k to correspond to a dot product in some other space (as for instance the SV algorithms described in this book), it is possible that even though k is not positive definite in general, it still gives rise to a positive definite Gram matrix K with respect to the training data at hand. In this case, Proposition 2.16 tells us that nothing will go wrong during training when we work with these data. Moreover, if k leads to a matrix with some small negative eigenvalues, we can add a small multiple of some strictly positive definite kernel k' (such as the identity $k'(x_i, x_j) = \delta_{ij}$) to obtain a positive definite matrix. To see this, suppose that $\lambda_{\min} < 0$ is the minimal eigenvalue of k's Gram matrix. Note that being strictly positive definite, the Gram matrix K' of k' satisfies

$$\min_{\|\boldsymbol{\alpha}\|=1} \langle \boldsymbol{\alpha}, K'\boldsymbol{\alpha} \rangle \geq \lambda'_{\min} > 0, \tag{2.65}$$

where $\lambda'_{\min}$ denotes its minimal eigenvalue, and the first inequality follows from Rayleigh's principle (B.57). Therefore, provided that $\lambda_{\min} + \lambda\lambda'_{\min} \geq 0$, we have

$$\langle \boldsymbol{\alpha}, (K + \lambda K')\boldsymbol{\alpha} \rangle = \langle \boldsymbol{\alpha}, K\boldsymbol{\alpha} \rangle + \lambda \langle \boldsymbol{\alpha}, K'\boldsymbol{\alpha} \rangle \geq \|\boldsymbol{\alpha}\|^2 \left(\lambda_{\min} + \lambda\lambda'_{\min}\right) \geq 0 \tag{2.66}$$

for all $\boldsymbol{\alpha} \in \mathbb{R}^m$, rendering $(K + \lambda K')$ positive definite.

2.3 Examples and Properties of Kernels

Polynomial

For the following examples, let us assume that $\mathcal{X} \subset \mathbb{R}^N$. Besides homogeneous polynomial kernels (cf. Proposition 2.1),

$$k(x, x') = \langle x, x' \rangle^d, \tag{2.67}$$

Gaussian

Boser, Guyon, and Vapnik [62, 223, 561] suggest the usage of Gaussian radial basis function kernels [26, 4],

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\,\sigma^2}\right), \tag{2.68}$$

Sigmoid

where $\sigma > 0$, and sigmoid kernels,

$$k(x, x') = \tanh(\kappa \langle x, x' \rangle + \vartheta), \tag{2.69}$$

where $\kappa > 0$ and $\vartheta < 0$. By applying Theorem 13.4 below, one can check that the latter kernel is not actually positive definite (see Section 4.6 and [85, 511] and the discussion in Example 4.25). Curiously, it has nevertheless successfully been used in practice. The reasons for this are discussed in [467].

Inhomogeneous Polynomial

Other useful kernels include the inhomogeneous polynomial,

$$k(x, x') = \left(\langle x, x' \rangle + c\right)^d, \tag{2.70}$$

($d \in \mathbb{N}, c \geq 0$) and the B_n-spline kernel [501, 572] (I_X denoting the indicator (or characteristic) function on the set X, and $\otimes$ the convolution operation, $(f \otimes g)(x) := \int f(x')g(x' - x)dx'$),

B_n-Spline of Odd Order

$$k(x, x') = B_{2p+1}(\|x - x'\|) \text{ with } B_n := \bigotimes_{i=1}^{n} I_{[-\frac{1}{2}, \frac{1}{2}]}. \tag{2.71}$$

The kernel computes B-splines of order $2p + 1$ ($p \in \mathbb{N}$), defined by the $(2p + 1)$-fold convolution of the unit interval $[-1/2, 1/2]$. See Section 4.4.1 for further details and a regularization theoretic analysis of this kernel.

Invariance of Kernels

Note that all these kernels have the convenient property of unitary invariance, $k(x, x') = k(Ux, Ux')$ if $U^\top = U^{-1}$, for instance if U is a rotation. If we consider complex numbers, then we have to use the adjoint $U^* := \overline{U}^\top$ instead of the transpose.

RBF Kernels

Radial basis function (RBF) kernels are kernels that can be written in the form

$$k(x, x') = f(d(x, x')), \tag{2.72}$$

where d is a metric on $\mathcal{X}$, and f is a function on $\mathbb{R}_0^+$. Examples thereof are the Gaussians and B-splines mentioned above. Usually, the metric arises from the dot product; $d(x, x') = \|x - x'\| = \sqrt{\langle x - x', x - x' \rangle}$. In this case, RBF kernels are unitary invariant, too. In addition, they are translation invariant; in other words, $k(x, x') = k(x + x_0, x' + x_0)$ for all $x_0 \in \mathcal{X}$.

In some cases, invariance properties alone can distinguish particular kernels: in Section 2.1, we explained how using polynomial kernels $\langle x, x' \rangle^d$ corresponds to mapping into a feature space whose dimensions are spanned by all possible dth order monomials in input coordinates. The different dimensions are scaled with the square root of the number of ordered products of the respective d entries (e.g., $\sqrt{2}$ in (2.13)). These scaling factors precisely ensure invariance under the group of all orthogonal transformations (rotations and mirroring operations). In many cases, this is a desirable property: it ensures that the results of a learning procedure do not depend on which orthonormal coordinate system (with fixed origin) we use for representing our input data.

Proposition 2.17 (Invariance of Polynomial Kernels [480]) *Up to a scaling factor, the kernel $k(x, x') = \langle x, x' \rangle^d$ is the only kernel inducing a map into a space of* all *monomials of degree d which is invariant under orthogonal transformations of $\mathbb{R}^N$.*

Properties of RBF Kernels

Some interesting additional structure exists in the case of a Gaussian RBF kernel k (2.68). As $k(x, x) = 1$ for all $x \in \mathcal{X}$, each mapped example has unit length, $\|\Phi(x)\| =$

1 (Problem 2.18 shows how to achieve this for general kernels). Moreover, as $k(x, x') > 0$ for all $x, x' \in \mathcal{X}$, all points lie inside the same orthant in feature space. To see this, recall that for unit length vectors, the dot product (1.3) equals the cosine of the enclosed angle. We obtain

$$\cos(\angle(\Phi(x), \Phi(x'))) = \langle \Phi(x), \Phi(x') \rangle = k(x, x') > 0, \tag{2.73}$$

which amounts to saying that the enclosed angle between any two mapped examples is smaller than $\pi/2$.

The above seems to indicate that in the Gaussian case, the mapped data lie in a fairly restricted area of feature space. However, in another sense, they occupy a space which is as large as possible:

Theorem 2.18 (Full Rank of Gaussian RBF Gram Matrices [360]) *Suppose that $x_1, \ldots, x_m \subset \mathcal{X}$ are distinct points, and $\sigma \neq 0$. The matrix K given by*

$$K_{ij} := \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) \tag{2.74}$$

has full rank.

In other words, the points $\Phi(x_1), \ldots, \Phi(x_m)$ are linearly independent (provided no two x_i are the same). They span an m-dimensional subspace of $\mathcal{H}$. Therefore a Gaussian kernel defined on a domain of infinite cardinality, with no a priori restriction on the number of training examples, produces a feature space of *infinite* dimension. Nevertheless, an analysis of the shape of the mapped data in feature space shows that capacity is distributed in a way that ensures smooth and simple estimates whenever possible (see Section 12.4).

Infinite-Dimensional Feature Space

The examples given above all apply to the case of vectorial data. Let us next give an example where $\mathcal{X}$ is *not* a vector space [42].

Proposition 2.19 (Similarity of Probabilistic Events) *If $(\mathcal{X}, \mathcal{C}, \mathrm{P})$ is a probability space with σ-algebra $\mathcal{C}$ and probability measure P, then*

$$k(A, B) = \mathrm{P}(A \cap B) - \mathrm{P}(A)\mathrm{P}(B) \tag{2.75}$$

is a positive definite kernel on $\mathcal{C} \times \mathcal{C}$.

Proof To see this, we define a feature map

$$\Phi : A \mapsto (I_A - \mathrm{P}(A)), \tag{2.76}$$

where I_A is the characteristic function on A. On the feature space, which consists of functions on $\mathcal{X}$ taking values in $[-1, 1]$, we use the dot product

$$\langle f, g \rangle := \int_{\mathcal{X}} f \cdot g \, d\mathrm{P}. \tag{2.77}$$

The result follows by noticing $\langle I_A, I_B \rangle = \mathrm{P}(A \cap B)$ and $\langle I_A, \mathrm{P}(B) \rangle = \mathrm{P}(A)\mathrm{P}(B)$.

■

Further examples include kernels for string matching, as proposed by [585, 234, 23]. We shall describe these, and address the general problem of designing kernel functions, in Chapter 13.

The next section will return to the connection between kernels and feature spaces. Readers who are eager to move on to SV algorithms may want to skip this section, which is somewhat more technical.

2.4 The Representation of Dissimilarities in Linear Spaces

2.4.1 Conditionally Positive Definite Kernels

We now proceed to a larger class of kernels than that of the positive definite ones. This larger class is interesting in several regards. First, it will turn out that some kernel algorithms work with this class, rather than only with positive definite kernels. Second, its relationship to positive definite kernels is a rather interesting one, and a number of connections between the two classes provide understanding of kernels in general. Third, they are intimately related to a question which is a variation on the central aspect of positive definite kernels: the latter can be thought of as dot products in feature spaces; the former, on the other hand, can be embedded as *distance measures* arising from norms in feature spaces.

The present section thus attempts to extend the utility of the kernel trick by looking at the problem of which kernels can be used to compute distances in feature spaces. The underlying mathematical results have been known for quite a while [465]; some of them have already attracted interest in the kernel methods community in various contexts [515, 234].

Clearly, the squared distance $\|\Phi(x) - \Phi(x')\|^2$ in the feature space associated with a pd kernel k can be computed, using $k(x, x') = \langle \Phi(x), \Phi(x') \rangle$, as

$$\|\Phi(x) - \Phi(x')\|^2 = k(x, x) + k(x', x') - 2k(x, x'). \tag{2.78}$$

Positive definite kernels are, however, not the full story: there exists a *larger* class of kernels that can be used as generalized distances, and the present section will describe why and how [468].

Let us start by considering how a dot product and the corresponding distance measure are affected by a translation of the data, $x \mapsto x - x_0$. Clearly, $\|x - x'\|^2$ is translation invariant while $\langle x, x' \rangle$ is not. A short calculation shows that the effect of the translation can be expressed in terms of $\|. - .\|^2$ as

$$\langle (x - x_0), (x' - x_0) \rangle = \frac{1}{2} \left(-\|x - x'\|^2 + \|x - x_0\|^2 + \|x_0 - x'\|^2 \right). \tag{2.79}$$

Note that this, just like $\langle x, x' \rangle$, is still a pd kernel: $\sum_{i,j} c_i c_j \langle (x_i - x_0), (x_j - x_0) \rangle = \| \sum_i c_i (x_i - x_0) \|^2 \geq 0$ holds true for any c_i. For any choice of $x_0 \in \mathcal{X}$, we thus get a similarity measure (2.79) associated with the dissimilarity measure $\|x - x'\|$.

This naturally leads to the question of whether (2.79) might suggest a connection

Further examples include kernels for string matching, as proposed by [585, 234, 23]. We shall describe these, and address the general problem of designing kernel functions, in Chapter 13.

The next section will return to the connection between kernels and feature spaces. Readers who are eager to move on to SV algorithms may want to skip this section, which is somewhat more technical.

2.4 The Representation of Dissimilarities in Linear Spaces

2.4.1 Conditionally Positive Definite Kernels

We now proceed to a larger class of kernels than that of the positive definite ones. This larger class is interesting in several regards. First, it will turn out that some kernel algorithms work with this class, rather than only with positive definite kernels. Second, its relationship to positive definite kernels is a rather interesting one, and a number of connections between the two classes provide understanding of kernels in general. Third, they are intimately related to a question which is a variation on the central aspect of positive definite kernels: the latter can be thought of as dot products in feature spaces; the former, on the other hand, can be embedded as *distance measures* arising from norms in feature spaces.

The present section thus attempts to extend the utility of the kernel trick by looking at the problem of which kernels can be used to compute distances in feature spaces. The underlying mathematical results have been known for quite a while [465]; some of them have already attracted interest in the kernel methods community in various contexts [515, 234].

Clearly, the squared distance $\|\Phi(x) - \Phi(x')\|^2$ in the feature space associated with a pd kernel k can be computed, using $k(x, x') = \langle \Phi(x), \Phi(x') \rangle$, as

$$\|\Phi(x) - \Phi(x')\|^2 = k(x, x) + k(x', x') - 2k(x, x'). \tag{2.78}$$

Positive definite kernels are, however, not the full story: there exists a *larger* class of kernels that can be used as generalized distances, and the present section will describe why and how [468].

Let us start by considering how a dot product and the corresponding distance measure are affected by a translation of the data, $x \mapsto x - x_0$. Clearly, $\|x - x'\|^2$ is translation invariant while $\langle x, x' \rangle$ is not. A short calculation shows that the effect of the translation can be expressed in terms of $\|. - .\|^2$ as

$$\langle (x - x_0), (x' - x_0) \rangle = \frac{1}{2} \left(-\|x - x'\|^2 + \|x - x_0\|^2 + \|x_0 - x'\|^2 \right). \tag{2.79}$$

Note that this, just like $\langle x, x' \rangle$, is still a pd kernel: $\sum_{i,j} c_i c_j \langle (x_i - x_0), (x_j - x_0) \rangle = \| \sum_i c_i (x_i - x_0) \|^2 \geq 0$ holds true for any c_i. For any choice of $x_0 \in \mathcal{X}$, we thus get a similarity measure (2.79) associated with the dissimilarity measure $\|x - x'\|$.

This naturally leads to the question of whether (2.79) might suggest a connection

that also holds true in more general cases: what kind of nonlinear dissimilarity measure do we have to substitute for $\|. - .\|^2$ on the right hand side of (2.79), to ensure that the left hand side becomes positive definite? To state the answer, we first need to define the appropriate class of kernels.

The following definition differs from Definition 2.4 only in the additional constraint on the sum of the c_i. Below, $\mathbb{K}$ is a shorthand for $\mathbb{C}$ or $\mathbb{R}$; the definitions are the same in both cases.

Definition 2.20 (Conditionally Positive Definite Matrix) *A symmetric $m \times m$ matrix K ($m \geq 2$) taking values in $\mathbb{K}$ and satisfying*

$$\sum_{i,j=1}^{m} c_i \bar{c}_j K_{ij} \geq 0 \text{ for all } c_i \in \mathbb{K}, \text{ with } \sum_{i=1}^{m} c_i = 0, \tag{2.80}$$

is called conditionally positive definite (cpd).

Definition 2.21 (Conditionally Positive Definite Kernel) *Let $\mathcal{X}$ be a nonempty set. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{K}$ which for all $m \geq 2, x_1, \ldots, x_m \in \mathcal{X}$ gives rise to a conditionally positive definite Gram matrix is called a* conditionally positive definite (cpd) kernel.

Note that symmetry is also required in the complex case. Due to the additional constraint on the coefficients c_i, it does not follow automatically anymore, as it did in the case of complex positive definite matrices and kernels. In Chapter 4, we will revisit cpd kernels. There, we will actually introduce cpd kernels of different orders. The definition given in the current chapter covers the case of kernels which are cpd *of order* 1.

Connection PD — CPD

Proposition 2.22 (Constructing PD Kernels from CPD Kernels [42]) *Let $x_0 \in \mathcal{X}$, and let k be a symmetric kernel on $\mathcal{X} \times \mathcal{X}$. Then*

$$\tilde{k}(x, x') := \frac{1}{2}(k(x, x') - k(x, x_0) - k(x_0, x') + k(x_0, x_0))$$

is positive definite if and only if k is conditionally positive definite.

The proof follows directly from the definitions and can be found in [42]. This result does generalize (2.79): the negative squared distance kernel is indeed cpd, since $\sum_i c_i = 0$ implies $-\sum_{i,j} c_i c_j \|x_i - x_j\|^2 = -\sum_i c_i \sum_j c_j \|x_j\|^2 - \sum_j c_j \sum_i c_i \|x_i\|^2 + 2\sum_{i,j} c_i c_j \langle x_i, x_j \rangle = 2\sum_{i,j} c_i c_j \langle x_i, x_j \rangle = 2\|\sum_i c_i x_i\|^2 \geq 0$. In fact, this implies that all kernels of the form

$$k(x, x') = -\|x - x'\|^\beta, 0 \leq \beta \leq 2 \tag{2.81}$$

are cpd (they are not pd),[10] by application of the following result (note that the case $\beta = 0$ is trivial):

10. Moreover, they are not cpd if $\beta > 2$ [42].

Proposition 2.23 (Fractional Powers and Logs of CPD Kernels [42]) *If $k : \mathcal{X} \times \mathcal{X} \to (-\infty, 0]$ is cpd, then so are $-(-k)^\alpha$ $(0 < \alpha < 1)$ and $-\ln(1-k)$.*

To state another class of cpd kernels that are not pd, note first that as a trivial consequence of Definition 2.20, we know that (i) sums of cpd kernels are cpd, and (ii) any constant $b \in \mathbb{R}$ is a cpd kernel. Therefore, any kernel of the form $k + b$, where k is cpd and $b \in \mathbb{R}$, is also cpd. In particular, since pd kernels are cpd, we can take any pd kernel and offset it by b, and it will still be at least cpd. For further examples of cpd kernels, cf. [42, 578, 205, 515].

2.4.2 Hilbert Space Representation of CPD Kernels

We now return to the main flow of the argument. Proposition 2.22 allows us to construct the feature map for k from that of the pd kernel $\tilde{k}$. To this end, fix $x_0 \in \mathcal{X}$ and define $\tilde{k}$ according to Proposition 2.22. Due to Proposition 2.22, $\tilde{k}$ is positive definite. Therefore, we may employ the Hilbert space representation $\Phi : \mathcal{X} \to \mathcal{H}$ of $\tilde{k}$ (cf. (2.32)), satisfying $\langle \Phi(x), \Phi(x') \rangle = \tilde{k}(x, x')$; hence,

$$\|\Phi(x) - \Phi(x')\|^2 = \tilde{k}(x, x) + \tilde{k}(x', x') - 2\tilde{k}(x, x'). \tag{2.82}$$

Substituting Proposition 2.22 yields

$$\|\Phi(x) - \Phi(x')\|^2 = -k(x, x') + \frac{1}{2}\left(k(x, x) + k(x', x')\right). \tag{2.83}$$

This implies the following result [465, 42].

Feature Map for CPD Kernels

Proposition 2.24 (Hilbert Space Representation of CPD Kernels) *Let k be a real-valued CPD kernel on $\mathcal{X}$, satisfying $k(x, x) = 0$ for all $x \in \mathcal{X}$. Then there exists a Hilbert space $\mathcal{H}$ of real-valued functions on $\mathcal{X}$, and a mapping $\Phi : \mathcal{X} \to \mathcal{H}$, such that*

$$\|\Phi(x) - \Phi(x')\|^2 = -k(x, x'). \tag{2.84}$$

If we drop the assumption $k(x, x) = 0$, the Hilbert space representation reads

$$\|\Phi(x) - \Phi(x')\|^2 = -k(x, x') + \frac{1}{2}\left(k(x, x) + k(x', x')\right). \tag{2.85}$$

It can be shown that if $k(x, x) = 0$ for all $x \in \mathcal{X}$, then

$$d(x, x') := \sqrt{-k(x, x')} = \|\Phi(x) - \Phi(x')\| \tag{2.86}$$

is a semi-metric: clearly, it is nonnegative and symmetric; additionally, it satisfies the triangle inequality, as can be seen by computing $d(x, x') + d(x', x'') = \|\Phi(x) - \Phi(x')\| + \|\Phi(x') - \Phi(x'')\| \geq \|\Phi(x) - \Phi(x'')\| = d(x, x'')$ [42].

It is a metric if $k(x, x') \neq 0$ for $x \neq x'$. We thus see that we can rightly think of k as the negative of a distance measure.

We next show how to represent *general* symmetric kernels (thus in particular cpd kernels) as symmetric bilinear forms Q in feature spaces. This generalization of the previously known feature space representation for pd kernels comes at a

cost: Q will no longer be a dot product. For our purposes, we can get away with this. The result will give us an intuitive understanding of Proposition 2.22: we can then write $\tilde{k}$ as $\tilde{k}(x, x') := Q(\Phi(x) - \Phi(x_0), \Phi(x') - \Phi(x_0))$. Proposition 2.22 thus essentially adds an origin in feature space which corresponds to the image $\Phi(x_0)$ of one point x_0 under the feature map.

Feature Map for General Symmetric Kernels

Proposition 2.25 (Vector Space Representation of Symmetric Kernels) *Let k be a real-valued symmetric kernel on $\mathcal{X}$. Then there exists a linear space $\mathcal{H}$ of real-valued functions on $\mathcal{X}$, endowed with a symmetric bilinear form $Q(.,.)$, and a mapping $\Phi : \mathcal{X} \to \mathcal{H}$, such that $k(x, x') = Q(\Phi(x), \Phi(x'))$.*

Proof The proof is a direct modification of the pd case. We use the map (2.21) and linearly complete the image as in (2.22). Define $Q(f, g) := \sum_{i=1}^{m} \sum_{j=1}^{m'} \alpha_i \beta_j k(x_i, x'_j)$. To see that it is well-defined, although it explicitly contains the expansion coefficients (which need not be unique), note that $Q(f, g) = \sum_{j=1}^{m'} \beta_j f(x'_j)$, independent of the α_i. Similarly, for g, note that $Q(f, g) = \sum_i \alpha_i g(x_i)$, hence it is independent of β_j. The last two equations also show that Q is bilinear; clearly, it is symmetric. ■

Note, moreover, that by definition of Q, k is a reproducing kernel for the feature space (which is not a Hilbert space): for all functions f (2.22), we have $Q(k(., x), f) = f(x)$; in particular, $Q(k(., x), k(., x')) = k(x, x')$.

Rewriting $\tilde{k}$ as $\tilde{k}(x, x') := Q(\Phi(x) - \Phi(x_0), \Phi(x') - \Phi(x_0))$ suggests an immediate generalization of Proposition 2.22: in practice, we might want to choose other points as origins in feature space — points that do not have a pre-image x_0 in the input domain, such as the mean of a set of points (cf. [543]). This will be useful when considering kernel PCA. It is only crucial that the behavior of our reference point under translation is identical to that of individual points. This is taken care of by the constraint on the sum of the c_i in the following proposition.

Matrix Centering

Proposition 2.26 (Exercise 2.23 in [42]) *Let K be a symmetric matrix, $\mathbf{e} \in \mathbb{R}^m$ be the vector of all ones, $\mathbf{1}$ the $m \times m$ identity matrix, and let $\mathbf{c} \in \mathbb{C}^m$ satisfy $\mathbf{e}^*\mathbf{c} = 1$. Then*

$$\tilde{K} := (\mathbf{1} - \mathbf{e}\mathbf{c}^*)K(\mathbf{1} - \mathbf{c}\mathbf{e}^*) \tag{2.87}$$

is positive definite if and only if K is conditionally positive definite.[11]

Proof "$\Longrightarrow$": suppose $\tilde{K}$ is positive definite. Thus for any $\mathbf{a} \in \mathbb{C}^m$ which satisfies $\mathbf{a}^*\mathbf{e} = \mathbf{e}^*\mathbf{a} = 0$, we have $0 \leq \mathbf{a}^*\tilde{K}\mathbf{a} = \mathbf{a}^*K\mathbf{a} + \mathbf{a}^*\mathbf{e}\mathbf{c}^*K\mathbf{c}\mathbf{e}^*\mathbf{a} - \mathbf{a}^*K\mathbf{c}\mathbf{e}^*\mathbf{a} - \mathbf{a}^*\mathbf{e}\mathbf{c}^*K\mathbf{a} = \mathbf{a}^*K\mathbf{a}$. This means that $0 \leq \mathbf{a}^*K\mathbf{a}$, proving that K is conditionally positive definite.

"$\Longleftarrow$": suppose K is conditionally positive definite. This means that we have to show that $\mathbf{a}^*\tilde{K}\mathbf{a} \geq 0$ for all $\mathbf{a} \in \mathbb{C}^m$. We have

$$\mathbf{a}^*\tilde{K}\mathbf{a} = \mathbf{a}^*(\mathbf{1} - \mathbf{e}\mathbf{c}^*)K(\mathbf{1} - \mathbf{c}\mathbf{e}^*)\mathbf{a} = \mathbf{s}^*K\mathbf{s} \text{ for } \mathbf{s} = (\mathbf{1} - \mathbf{c}\mathbf{e}^*)\mathbf{a}. \tag{2.88}$$

11. $\mathbf{c}^*$ is the vector obtained by transposing and taking the complex conjugate of $\mathbf{c}$.

All we need to show is $\mathbf{e}^*\mathbf{s} = 0$, since then we can use the fact that K is cpd to obtain $\mathbf{s}^* K\mathbf{s} \geq 0$. This can be seen as follows $\mathbf{e}^*\mathbf{s} = \mathbf{e}^*(\mathbf{1} - \mathbf{c}\mathbf{e}^*)\mathbf{a} = (\mathbf{e}^* - (\mathbf{e}^*\mathbf{c})\mathbf{e}^*)\mathbf{a} = (\mathbf{e}^* - \mathbf{e}^*)\mathbf{a} = 0$. ■

This result directly implies a corresponding generalization of Proposition 2.22:

Kernel Centering

Proposition 2.27 (Adding a General Origin) *Let k be a symmetric kernel, $x_1, \ldots, x_m \in \mathcal{X}$, and let $c_i \in \mathbb{C}$ satisfy $\sum_{i=1}^m c_i = 1$. Then*

$$\tilde{k}(x, x') := \frac{1}{2}\Big(k(x, x') - \sum_{i=1}^m c_i k(x, x_i) - \sum_{i=1}^m c_i k(x_i, x') + \sum_{i,j=1}^m c_i c_j k(x_i, x_j)\Big)$$

is positive definite if and only if k is conditionally positive definite.

Proof Consider a set of $m' \in \mathbb{N}$ points $x'_1, \ldots, x'_{m'} \in \mathcal{X}$, and let K be the $(m + m') \times (m + m')$ Gram matrix based on $x_1, \ldots, x_m, x'_1, \ldots, x'_{m'}$. Apply Proposition 2.26 using $c_{m+1} = \ldots = c_{m+m'} = 0$. ■

Application to SVMs

The above results show that conditionally positive definite kernels are a natural choice whenever we are dealing with a translation invariant problem, such as the SVM: maximization of the margin of separation between two classes of data is independent of the position of the origin. Seen in this light, it is not surprising that the structure of the dual optimization problem (cf. [561]) allows cpd kernels: as noted in [515, 507], the constraint $\sum_{i=1}^m \alpha_i y_i = 0$ projects out the same subspace as (2.80) in the definition of conditionally positive definite matrices.

Application to Kernel PCA

Another example of a kernel algorithm that works with conditionally positive definite kernels is Kernel PCA (Chapter 14), where the data are centered, thus removing the dependence on the origin in feature space. Formally, this follows from Proposition 2.26 for $c_i = 1/m$.

Application to Parzen Windows Classifiers

Let us consider another example. One of the simplest distance-based classification algorithms proceeds as follows. Given m_+ points labelled with $+1$, m_- points labelled with -1, and a mapped test point $\Phi(x)$, we compute the mean squared distances between the latter and the two classes, and assign it to the one for which this mean is smaller;

$$y = \operatorname{sgn}\left(\frac{1}{m_-}\sum_{y_i=-1}\|\Phi(x) - \Phi(x_i)\|^2 - \frac{1}{m_+}\sum_{y_i=1}\|\Phi(x) - \Phi(x_i)\|^2\right). \tag{2.89}$$

We use the distance kernel trick (Proposition 2.24) to express the decision function as a kernel expansion in the input domain: a short calculation shows that

$$y = \operatorname{sgn}\left(\frac{1}{m_+}\sum_{y_i=1}k(x, x_i) - \frac{1}{m_-}\sum_{y_i=-1}k(x, x_i) + b\right), \tag{2.90}$$

with the constant offset

$$b = \frac{1}{2m_-}\sum_{y_i=-1}k(x_i, x_i) - \frac{1}{2m_+}\sum_{y_i=1}k(x_i, x_i). \tag{2.91}$$

Note that for some cpd kernels, such as (2.81), $k(x_i, x_i)$ is always 0, and thus $b = 0$. For others, such as the commonly used Gaussian kernel, $k(x_i, x_i)$ is a nonzero constant, in which case b vanishes provided that $m_+ = m_-$. For normalized Gaussians, the resulting decision boundary can be interpreted as the Bayes decision based on two Parzen window density estimates of the classes; for general cpd kernels, the analogy is merely a formal one; that is, the decision functions take the same form.

Properties of CPD Kernels

Many properties of positive definite kernels carry over to the more general case of conditionally positive definite kernels, such as Proposition 13.1.

Using Proposition 2.22, one can prove an interesting connection between the two classes of kernels:

Proposition 2.28 (Connection PD — CPD [465]) *A kernel k is conditionally positive definite if and only if* $\exp(tk)$ *is positive definite for all $t > 0$.*

Positive definite kernels of the form $\exp(tk)$ ($t > 0$) have the interesting property that their nth root ($n \in \mathbb{N}$) is again a positive definite kernel. Such kernels are called *infinitely divisible*. One can show that, disregarding some technicalities, the logarithm of an infinitely divisible positive definite kernel mapping into $\mathbb{R}_0^+$ is a conditionally positive definite kernel.

2.4.3 Higher Order CPD Kernels

For the sake of completeness, we now present some material which is of interest to one section later in the book (Section 4.8), but not central for the present chapter. We follow [341, 204].

Definition 2.29 (Conditionally Positive Definite Functions of Order q) *A continuous function h, defined on* $[0, \infty)$, *is called conditionally positive definite (cpd) of order q on $\mathbb{R}^N$ if for any distinct points $x_1, \ldots, x_m \in \mathbb{R}^N$, the quadratic form,*

$$\sum_{i,j=1}^{m} \alpha_i \alpha_j h(\|x_i - x_j\|^2), \tag{2.92}$$

is nonnegative, provided that the scalars $\alpha_1, \ldots, \alpha_m$ satisfy $\sum_{i=1}^m \alpha_i p(x_i) = 0$, for all polynomials $p(\cdot)$ on $\mathbb{R}^N$ of degree lower than q.

Let Π_q^N denote the space of polynomials of degree lower than q on $\mathbb{R}^N$. By definition, every cpd function h of order q generates a positive definite kernel for SV expansions in the space of functions orthogonal to Π_q^N, by setting $k(x, x') := h(\|x - x'\|^2)$.

There exists also an analogue to the positive definiteness of the integral operator in the conditions of Mercer's theorem. In [157, 341] it is shown that for cpd functions h of order q, we have

$$\int h(\|x - x'\|^2) f(x) f(x') dx dx' \geq 0, \tag{2.93}$$

provided that the projection of f onto Π_q^N is zero.

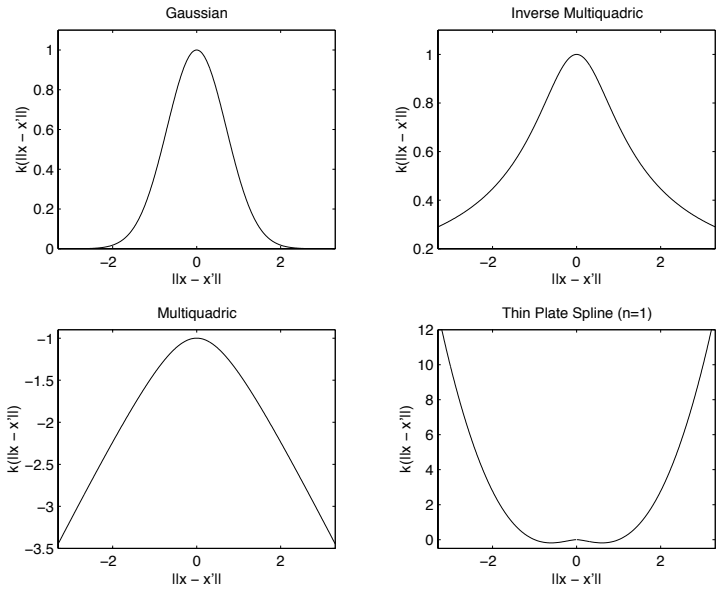


Figure 2.4 Conditionally positive definite functions, as described in Table 2.1. Where applicable, we set the free parameter c to 1; β is set to 2. Note that cpd kernels need not be positive anywhere (e.g., the Multiquadric kernel).

Table 2.1 Examples of Conditionally Positive Definite Kernels. The fact that the exponential kernel is pd (i.e., cpd of order 0) follows from (2.81) and Proposition 2.28.

Kernel	Order	
$e^{-c\|x-x'\|^\beta}$, $0 \leq \beta \leq 2$	0	Exponential
$\frac{1}{\sqrt{\|x-x'\|^2+c^2}}$	0	Inverse Multiquadric
$-\sqrt{\|x-x'\|^2+c^2}$	1	Multiquadric
$\|x-x'\|^{2n} \ln \|x-x'\|$	n	Thin Plate Spline

Definition 2.30 (Completely Monotonic Functions) *A function $h(x)$ is called completely monotonic of order q if*

$$(-1)^n \frac{d^n}{dx^n} h(x) \geq 0 \text{ for all } x \in [0, \infty) \text{ and } n \geq q. \tag{2.94}$$

It can be shown [464, 465, 360] that a function $h(x^2)$ is conditionally positive definite if and only if $h(x)$ is completely monotonic of the same order. This gives a (sometimes simpler) criterion for checking whether a function is cpd or not.

If we use cpd kernels in learning algorithms, we must ensure orthogonality of the estimate with respect to Π_q^N. This is usually done via constraints $\sum_{i=1}^m \alpha_i p(x_i) = 0$ for all polynomials $p(\cdot)$ on $\mathbb{R}^N$ of degree lower than q (see Section 4.8).

2.5 Summary

The crucial ingredient of SVMs and other kernel methods is the so-called kernel trick (see (2.7) and Remark 2.8), which permits the computation of dot products in high-dimensional feature spaces, using simple functions defined on pairs of input patterns. This trick allows the formulation of nonlinear variants of any algorithm that can be cast in terms of dot products, SVMs being but the most prominent example. The mathematical result underlying the kernel trick is almost a century old [359]. Nevertheless, it was only much later that it was exploited by the machine learning community for the analysis [4] and construction of algorithms [62], and that it was described as a general method for constructing nonlinear generalizations of dot product algorithms [480].

The present chapter has reviewed the mathematical theory of kernels. We started with the class of polynomial kernels, which can be motivated as computing a combinatorially large number of monomial features rather efficiently. This led to the general question of which kernel can be used, or: which kernel can be represented as a dot product in a linear feature space. We defined this class and discussed some of its properties. We described several ways how, given such a kernel, one can construct a representation in a feature space. The most well-known representation employs Mercer's theorem, and represents the feature space as an ℓ_2 space defined in terms of the eigenfunctions of an integral operator associated with the kernel. An alternative representation uses elements of the theory of reproducing kernel Hilbert spaces, and yields additional insights, representing the linear space as a space of functions written as kernel expansions. We gave an in-depth discussion of the kernel trick in its general form, including the case where we are interested in dissimilarities rather than similarities; that is, when we want to come up with nonlinear generalizations of distance-based algorithms rather than dot-product-based algorithms.

In both cases, the underlying philosophy is the same: we are trying to express a complex nonlinear algorithm in terms of simple geometrical concepts, and we are then dealing with it in a linear space. This linear space may not always be readily available; in some cases, it may even be hard to construct explicitly. Nevertheless, for the sake of design and analysis of the algorithms, it is sufficient to know that the linear space exists, empowering us to use the full potential of geometry, linear algebra and functional analysis.

2.6 Problems

2.1 (Monomial Features in $\mathbb{R}^2$ •) *Verify the second equality in (2.9).*

2.2 (Multiplicity of Monomial Features in $\mathbb{R}^N$ [515] ••) *Consider the monomial kernel $k(x, x') = \langle x, x' \rangle^d$ (where $x, x' \in \mathbb{R}^N$), generating monomial features of order d. Prove*

2.5 Summary

The crucial ingredient of SVMs and other kernel methods is the so-called kernel trick (see (2.7) and Remark 2.8), which permits the computation of dot products in high-dimensional feature spaces, using simple functions defined on pairs of input patterns. This trick allows the formulation of nonlinear variants of any algorithm that can be cast in terms of dot products, SVMs being but the most prominent example. The mathematical result underlying the kernel trick is almost a century old [359]. Nevertheless, it was only much later that it was exploited by the machine learning community for the analysis [4] and construction of algorithms [62], and that it was described as a general method for constructing nonlinear generalizations of dot product algorithms [480].

The present chapter has reviewed the mathematical theory of kernels. We started with the class of polynomial kernels, which can be motivated as computing a combinatorially large number of monomial features rather efficiently. This led to the general question of which kernel can be used, or: which kernel can be represented as a dot product in a linear feature space. We defined this class and discussed some of its properties. We described several ways how, given such a kernel, one can construct a representation in a feature space. The most well-known representation employs Mercer's theorem, and represents the feature space as an ℓ_2 space defined in terms of the eigenfunctions of an integral operator associated with the kernel. An alternative representation uses elements of the theory of reproducing kernel Hilbert spaces, and yields additional insights, representing the linear space as a space of functions written as kernel expansions. We gave an in-depth discussion of the kernel trick in its general form, including the case where we are interested in dissimilarities rather than similarities; that is, when we want to come up with nonlinear generalizations of distance-based algorithms rather than dot-product-based algorithms.

In both cases, the underlying philosophy is the same: we are trying to express a complex nonlinear algorithm in terms of simple geometrical concepts, and we are then dealing with it in a linear space. This linear space may not always be readily available; in some cases, it may even be hard to construct explicitly. Nevertheless, for the sake of design and analysis of the algorithms, it is sufficient to know that the linear space exists, empowering us to use the full potential of geometry, linear algebra and functional analysis.

2.6 Problems

2.1 (Monomial Features in $\mathbb{R}^2$ •) *Verify the second equality in (2.9).*

2.2 (Multiplicity of Monomial Features in $\mathbb{R}^N$ [515] ••) *Consider the monomial kernel $k(x, x') = \langle x, x' \rangle^d$ (where $x, x' \in \mathbb{R}^N$), generating monomial features of order d. Prove*

2.5 Summary

The crucial ingredient of SVMs and other kernel methods is the so-called kernel trick (see (2.7) and Remark 2.8), which permits the computation of dot products in high-dimensional feature spaces, using simple functions defined on pairs of input patterns. This trick allows the formulation of nonlinear variants of any algorithm that can be cast in terms of dot products, SVMs being but the most prominent example. The mathematical result underlying the kernel trick is almost a century old [359]. Nevertheless, it was only much later that it was exploited by the machine learning community for the analysis [4] and construction of algorithms [62], and that it was described as a general method for constructing nonlinear generalizations of dot product algorithms [480].

The present chapter has reviewed the mathematical theory of kernels. We started with the class of polynomial kernels, which can be motivated as computing a combinatorially large number of monomial features rather efficiently. This led to the general question of which kernel can be used, or: which kernel can be represented as a dot product in a linear feature space. We defined this class and discussed some of its properties. We described several ways how, given such a kernel, one can construct a representation in a feature space. The most well-known representation employs Mercer's theorem, and represents the feature space as an ℓ_2 space defined in terms of the eigenfunctions of an integral operator associated with the kernel. An alternative representation uses elements of the theory of reproducing kernel Hilbert spaces, and yields additional insights, representing the linear space as a space of functions written as kernel expansions. We gave an in-depth discussion of the kernel trick in its general form, including the case where we are interested in dissimilarities rather than similarities; that is, when we want to come up with nonlinear generalizations of distance-based algorithms rather than dot-product-based algorithms.

In both cases, the underlying philosophy is the same: we are trying to express a complex nonlinear algorithm in terms of simple geometrical concepts, and we are then dealing with it in a linear space. This linear space may not always be readily available; in some cases, it may even be hard to construct explicitly. Nevertheless, for the sake of design and analysis of the algorithms, it is sufficient to know that the linear space exists, empowering us to use the full potential of geometry, linear algebra and functional analysis.

2.6 Problems

2.1 (Monomial Features in $\mathbb{R}^2$ •) *Verify the second equality in (2.9).*

2.2 (Multiplicity of Monomial Features in $\mathbb{R}^N$ [515] ••) *Consider the monomial kernel $k(x, x') = \langle x, x' \rangle^d$ (where $x, x' \in \mathbb{R}^N$), generating monomial features of order d. Prove*

that a valid feature map for this kernel can be defined coordinate-wise as

$$\Phi_{\mathbf{m}}(\mathbf{x}) = \sqrt{\frac{d!}{\prod_{i=1}^{n}[\mathbf{m}]_i!}} \prod_{i=1}^{n} [\mathbf{x}]_i^{[\mathbf{m}]_i} \tag{2.95}$$

for every $\mathbf{m} \in \mathbb{N}^n$, $\sum_{i=1}^{n}[\mathbf{m}]_i = d$ *(i.e., every such* $\mathbf{m}$ *corresponds to one dimension of* $\mathcal{H}$*).*

2.3 (Inhomogeneous Polynomial Kernel ••) *Prove that the kernel (2.70) induces a feature map into the space of all monomials up to degree d. Discuss the role of c.*

2.4 (Eigenvalue Criterion of Positive Definiteness •) *Prove that a symmetric matrix is positive definite if and only if all its eigenvalues are nonnegative (see Appendix B).*

2.5 (Dot Products are Kernels •) *Prove that dot products (Definition B.7) are positive definite kernels.*

2.6 (Kernels on Finite Domains ••) *Prove that for finite* $\mathcal{X}$*, say* $\mathcal{X} = \{x_1, \ldots, x_m\}$*, k is a kernel if and only if the* $m \times m$ *matrix* $(k(x_i, x_j))_{ij}$ *is positive definite.*

2.7 (Positivity on the Diagonal •) *From Definition 2.5, prove that a kernel satisfies* $k(x, x) \geq 0$ *for all* $x \in \mathcal{X}$*.*

2.8 (Cauchy-Schwarz for Kernels ••) *Give an elementary proof of Proposition 2.7.*

Hint: start with the general form of a symmetric 2×2 *matrix, and derive conditions for its coefficients that ensure that it is positive definite.*

2.9 (PD Kernels Vanishing on the Diagonal •) *Use Proposition 2.7 to prove that a kernel satisfying* $k(x, x) =$ *for all* $x \in \mathcal{X}$ *is identically zero.*

How does the RKHS look in this case? Hint: use (2.31).

2.10 (Two Kinds of Positivity •) *Give an example of a kernel which is positive definite according to Definition 2.5, but not positive in the sense that* $k(x, x') \geq 0$ *for all* x, x'*.*

Give an example of a kernel where the contrary is the case.

2.11 (General Coordinate Transformations •) *Prove that if* $\sigma : \mathcal{X} \to \mathcal{X}$ *is a bijection, and* $k(x, x')$ *is a kernel, then* $k(\sigma(x), \sigma(x'))$ *is a kernel, too.*

2.12 (Positivity on the Diagonal •) *Prove that positive definite kernels are positive on the diagonal,* $k(x, x) \geq 0$ *for all* $x \in \mathcal{X}$*. Hint: use* $m = 1$ *in (2.15).*

2.13 (Symmetry of Complex Kernels ••) *Prove that complex-valued positive definite kernels are symmetric (2.18).*

2.14 (Real Kernels vs. Complex Kernels •) *Prove that a real matrix satisfies (2.15) for all* $c_i \in \mathbb{C}$ *if and only if it is symmetric and it satisfies (2.15) for* real *coefficients* c_i*.*

Hint: decompose each c_i *in (2.15) into real and imaginary parts.*

2.15 (Rank-One Kernels •) *Prove that if f is a real-valued function on $\mathcal{X}$, then $k(x, x') := f(x)f(x')$ is a positive definite kernel.*

2.16 (Bayes Kernel ••) *Consider a binary pattern recognition problem. Specialize the last problem to the case where $f : \mathcal{X} \to \{\pm 1\}$ equals the Bayes decision function $y(x)$, i.e., the classification with minimal risk subject to an underlying distribution $P(x, y)$ generating the data.*

Argue that this kernel is particularly suitable since it renders the problem linearly separable in a 1D feature space: State a decision function (cf. (1.35)) that solves the problem (hint: you just need one parameter α, and you may set it to 1; moreover, use $b = 0$) [124].

The final part of the problem requires knowledge of Chapter 16: Consider now the situation where some prior $P(f)$ over the target function class is given. What would the optimal kernel be in this case? Discuss the connection to Gaussian processes.

2.17 (Inhomogeneous Polynomials •) *Prove that the inhomogeneous polynomial (2.70) is a positive definite kernel, e.g., by showing that it is a linear combination of homogeneous polynomial kernels with positive coefficients. What kind of features does this kernel compute [561]?*

2.18 (Normalization in Feature Space •) *Given a kernel k, construct a corresponding normalized kernel $\tilde{k}$ by normalizing the feature map $\tilde{\Phi}$ such that for all $x \in \mathcal{X}$, $\|\tilde{\Phi}(x)\| = 1$ (cf. also Definition 12.35). Discuss the relationship between normalization in input space and normalization in feature space for Gaussian kernels and homogeneous polynomial kernels.*

2.19 (Cosine Kernel •) *Suppose $\mathcal{X}$ is a dot product space, and $x, x' \in \mathcal{X}$. Prove that $k(x, x') = \cos(\angle(x, x))$ is a positive definite kernel. Hint: use Problem 2.18.*

2.20 (Alignment Kernel •) *Let $\langle K, K' \rangle_F := \sum_{ij} K_{ij} K'_{ij}$ be the Frobenius dot product of two matrices. Prove that the* empirical alignment *of two Gram matrices [124], $A(K, K') := \langle K, K' \rangle_F / \sqrt{\langle K, K \rangle_F \langle K', K' \rangle_F}$, is a positive definite kernel.*

Note that the alignment can be used for model selection, putting $K'_{ij} := y_i y_j$ (cf. Problem 2.16) and $K_{ij} := \operatorname{sgn}(k(x_i, x_j))$ or $K_{ij} := \operatorname{sgn}(k(x_i, x_j)) - b$ (cf. [124]).

2.21 (Equivalence Relations as Kernels •••) *Consider a similarity measure $k : \mathcal{X} \to \{0, 1\}$ with*

$$k(x, x) = 1 \text{ for all } x \in \mathcal{X}. \tag{2.96}$$

Prove that k is a positive definite kernel if and only if, for all $x, x', x'' \in \mathcal{X}$,

$$k(x, x') = 1 \iff k(x', x) = 1 \text{ and} \tag{2.97}$$

$$k(x, x') = k(x', x'') = 1 \implies k(x, x'') = 1. \tag{2.98}$$

Equations (2.96) to (2.98) amount to saying that $k = I_T$, where $T \subset \mathcal{X} \times \mathcal{X}$ is an equivalence relation.

As a simple example, consider an undirected graph, and let $(x, x') \in T$ whenever x and x' are in the same connected component of the graph. Show that T is an equivalence relation.

Find examples of equivalence relations that lend themselves to an interpretation as similarity measures. Discuss whether there are other relations that one might want to use as similarity measures.

2.22 (Different Feature Spaces for the Same Kernel •) *Give an example of a kernel with two valid feature maps Φ_1, Φ_2, mapping into spaces $\mathcal{H}_1, \mathcal{H}_2$ of different dimensions.*

2.23 (Converse of Mercer's Theorem •) *Prove that if an integral operator kernel k admits a uniformly convergent dot product representation on some compact set $\mathcal{X} \times \mathcal{X}$,*

$$k(x, x') = \sum_{i=1}^{\infty} \psi_i(x)\psi_i(x'), \tag{2.99}$$

then it is positive definite. Hint: show that

$$\int_{\mathcal{X}\times\mathcal{X}} \left(\sum_{i=1}^{\infty} \psi_i(x)\psi_i(x') \right) f(x)f(x')\,dx\,dx' = \sum_{i=1}^{\infty} \left(\int_{\mathcal{X}} \psi_i(x)f(x)\,dx \right)^2 \geq 0.$$

Argue that in particular, polynomial kernels (2.67) satisfy Mercer's conditions.

2.24 (∞-Norm of Mercer Eigenfunctions ••) *Prove that under the conditions of Theorem 2.10, we have, up to sets of measure zero,*

$$\sup_j \left\| \sqrt{\lambda_j}\psi_j \right\|_\infty \leq \sqrt{\|k\|_\infty} < \infty. \tag{2.100}$$

Hint: note that $\|k\|_\infty \geq k(x, x)$ up to sets of measures zero, and use the series expansion given in Theorem 2.10. Show, moreover, that it is not generally the case that

$$\sup_j \|\psi_j\|_\infty < \infty. \tag{2.101}$$

Hint: consider the case where $\mathcal{X} = \mathbb{N}$, $\mu(\{n\}) := 2^{-n}$, and $k(i, j) := \delta_{ij}$. Show that

1. *$T_k((a_j)) = (a_j 2^{-j})$ for $(a_j) \in L_2(\mathcal{X}, \mu)$,*
2. *T_k satisfies $\langle (a_j), T_k(a_j) \rangle = \sum_j (a_j 2^{-j})^2 \geq 0$ and is thus positive definite,*
3. *$\lambda_j = 2^{-j}$ and $\psi_j = 2^{j/2} e_j$ form an orthonormal eigenvector decomposition of T_k (here, e_j is the jth canonical unit vector in ℓ_2), and*
4. *$\|\psi_j\|_\infty = 2^{j/2} = \lambda_j^{-1/2}$.*

Argue that the last statement shows that (2.101) is wrong and (2.100) is tight.[12]

2.25 (Generalized Feature Maps •••) *Via (2.38), Mercer kernels induce compact (integral) operators. Can you generalize the idea of defining a feature map associated with an*

12. Thanks to S. Smale and I. Steinwart for this exercise.

operator to more general bounded positive definite operators T? Hint: use the multiplication operator representation of T [467].

2.26 (Nyström Approximation (cf. [603]) •) *Consider the integral operator obtained by substituting the distribution* P *underlying the data into (2.38), i.e.,*

$$(T_k f)(x) = \int_{\mathcal{X}} k(x, x') f(x)\, d\mathrm{P}(x). \tag{2.102}$$

If the conditions of Mercer's theorem are satisfied, then k can be diagonalized as

$$k(x, x') = \sum_{j=1}^{N_{\mathcal{H}}} \lambda_j \psi_j(x) \psi_j(x'), \tag{2.103}$$

where λ_j and ψ_j satisfy the eigenvalue equation

$$\int_{\mathcal{X}} k(x, x') \psi_j(x)\, d\mathrm{P}(x) = \lambda_j \psi_j(x') \tag{2.104}$$

and the orthonormality conditions

$$\int_{\mathcal{X}} \psi_i(x) \psi_j(x) d\mathrm{P}(x) = \delta_{ij}. \tag{2.105}$$

Show that by replacing the integral by a summation over an iid sample $X = \{x_1, \ldots, x_m\}$ from P(x)*, one can recover the kernel PCA eigenvalue problem (Section 1.7). Hint: Start by evaluating (2.104) for $x' \in X$, to obtain m equations. Next, approximate the integral by a sum over the points in X, replacing $\int_{\mathcal{X}} k(x, x') \psi_j(x)\, d\mathrm{P}(x)$ by $\frac{1}{m} \sum_{n=1}^{m} k(x_n, x') \psi_j(x_n)$.*

Derive the orthogonality condition for the eigenvectors $(\psi_j(x_n))_{n=1,\ldots,m}$ from (2.105).

2.27 (Lorentzian Feature Spaces ••) *If a finite number of eigenvalues is negative, the expansion in Theorem 2.10 is still valid. Show that in this case, k corresponds to a Lorentzian symmetric bilinear form in a space with indefinite signature [467].*

Discuss whether this causes problems for learning algorithms utilizing these kernels. In particular, consider the cases of SV machines (Chapter 7) and Kernel PCA (Chapter 14).

2.28 (Symmetry of Reproducing Kernels •) *Show that reproducing kernels (Definition 2.9) are symmetric. Hint: use (2.35) and exploit the symmetry of the dot product.*

2.29 (Coordinate Representation in the RKHS ••) *Write $\langle \cdot, \cdot \rangle$ as a dot product of coordinate vectors by expressing the functions of the RKHS in the basis $(\sqrt{\lambda_n}\psi_n)_{n=1,\ldots,N_{\mathcal{H}}}$, which is orthonormal with respect to $\langle \cdot, \cdot \rangle$, i.e.,*

$$f(x) = \sum_{n=1}^{N_{\mathcal{H}}} \alpha_n \sqrt{\lambda_n} \psi_n(x). \tag{2.106}$$

Obtain an expression for the coordinates α_n, using (2.47) and $\alpha_n = \left\langle f, \sqrt{\lambda_n}\psi_n \right\rangle$. Show that $\mathcal{H}$ has the structure of a RKHS in the sense that for f and g given by (2.106), and

$$g(x) = \sum_{j=1}^{N_{\mathcal{H}}} \beta_j \sqrt{\lambda_j} \psi_j(x), \tag{2.107}$$

we have $\langle \boldsymbol{\alpha}, \boldsymbol{\beta} \rangle = \langle f, g \rangle$ *. Show, moreover, that* $f(x) = \langle \boldsymbol{\alpha}, \Phi(x) \rangle$ *in* $\mathcal{H}$*. In other words,* $\Phi(x)$ *is the coordinate representation of the kernel as a function of one argument.*

2.30 (Equivalence of Regularization Terms •) *Using (2.36) and (2.41), prove that* $\|\mathbf{w}\|^2$*, where* $\mathbf{w} = \sum_{i=1}^{m} \alpha_i \Phi(x_i)$*, is the same no matter whether* Φ *denotes the RKHS feature map (2.21) or the Mercer feature map (2.40).*

2.31 (Approximate Inversion of Gram Matrices ••) *Use the kernel PCA map (2.59) to derive a method for approximately inverting a large Gram matrix.*

2.32 (Effective Dimension of Feature Space •) *Building on Section 2.2.7, argue that for a finite data set, we are always effectively working in a finite-dimensional feature space.*

2.33 (Translation of a Dot Product •) *Prove (2.79).*

2.34 (Example of a CPD Kernel ••) *Argue that the hyperbolic tangent kernel (2.69) is effectively conditionally positive definite, if the input values are suitably restricted, since it can be approximated by* $k + b$*, where* k *is a polynomial kernel (2.67) and* $b \in \mathbb{R}$*. Discuss how this explains that hyperbolic tangent kernels can be used for SVMs although, as pointed out in number of works (e.g., [86], cf. the remark following (2.69)), they are not positive definite.*

2.35 (Polarization Identity ••) *Prove the* polarization identity*, stating that for any symmetric bilinear form* $\langle \cdot, \cdot \rangle : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$*, we have, for all* $x, x' \in \mathcal{X}$*,*

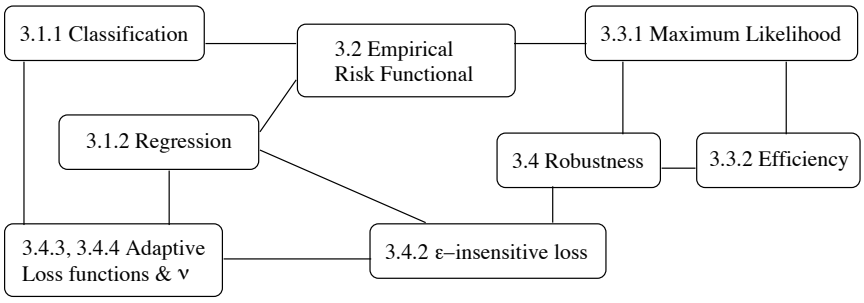
$$\langle x, x' \rangle = \frac{1}{4} \left(\langle x + x', x + x' \rangle - \langle x - x', x - x' \rangle \right). \tag{2.108}$$

Now consider the special case where $\langle \cdot, \cdot \rangle$ *is a Euclidean dot product and* $\langle x - x', x - x' \rangle$ *is the squared Euclidean distance between* x *and* x'*. Discuss why the polarization identity does not imply that the value of the dot product can be recovered from the distances alone. What else does one need?*

2.36 (Vector Space Representation of CPD Kernels •••) *Specialize the vector space representation of symmetric kernels (Proposition 2.25) to the case of cpd kernels. Can you identify a subspace on which a cpd kernel is actually pd?*

2.37 (Parzen Windows Classifiers in Feature Space ••) *Assume that* k *is a positive definite kernel. Compare the algorithm described in Section 1.2 with the one of (2.89). Construct situations where the two algorithms give different results. Hint: consider datasets where the class means coincide.*

2.38 (Canonical Distortion Kernel ○○○) *Can you define a kernel based on Baxter's canonical distortion metric [28]?*



Prerequisites

As usual, exercises for all sections can be found at the end. The chapter requires knowledge of probability theory, as introduced in Section B.1.

3.1 Loss Functions

Let us begin with a formal definition of what we mean by the loss incurred by a function f at location x, given an observation y.

Definition 3.1 (Loss Function) *Denote by $(x, y, f(x)) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{Y}$ the triplet consisting of a pattern x, an observation y and a prediction $f(x)$. Then the map $c : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$ with the property $c(x, y, y) = 0$ for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ will be called a loss function.*

Note that we require c to be a nonnegative function. This means that we will never get a payoff from an extra good prediction. If the latter was the case, we could always recover non-negativity (provided the loss is bounded from below), by using a simple shift operation (possibly depending on x). Likewise we can always satisfy the condition that exact predictions ($f(x) = y$) never cause any loss. The advantage of these extra conditions on c is that we know that the minimum of the loss is 0 and that it is obtainable, at least for a given x, y.

Minimized Loss $\neq$ Incurred Loss

Next we will formalize different kinds of *loss*, as described informally in the introduction of the chapter. Note that the incurred loss is not always the quantity that we will attempt to minimize. For instance, for algorithmic reasons, some loss functions will prove to be infeasible (the binary loss, for instance, can lead to NP-hard optimization problems [367]). Furthermore, statistical considerations such as the desire to obtain confidence levels on the prediction (Section 3.3.1) will also influence our choice.

3.1.1 Binary Classification

Misclassification Error

The simplest case to consider involves counting the misclassification error if pattern x is classified wrongly we incur loss 1, otherwise there is no penalty.:

$$c(x, y, f(x)) = \begin{cases} 0 & \text{if } y = f(x) \\ 1 & \text{otherwise} \end{cases} \tag{3.1}$$

This definition of c does not distinguish between different classes and types of errors (false positive or negative).[1]

Asymmetric and Input-Dependent Loss

A slight extension takes the latter into account. For the sake of simplicity let us assume, as in (3.1), that we have a binary classification problem. This time, however, the loss may depend on a function $\tilde{c}(x)$ which accounts for input-dependence, i.e.

$$c(x, y, f(x)) = \begin{cases} 0 & \text{if } y = f(x) \\ \tilde{c}(x) & \text{otherwise} \end{cases} \tag{3.2}$$

A simple (albeit slightly contrived) example is the classification of objects into rocks and diamonds. Clearly, the incurred loss will depend largely on the weight of the object under consideration.

Analogously, we might distinguish between errors for $y = 1$ and $y = -1$ (see, e.g., [331] for details). For instance, in a fraud detection application, we would like to be really sure about the situation before taking any measures, rather than losing potential customers. On the other hand, a blood bank should consider even the slightest suspicion of disease before accepting a donor.

Confidence Level

Rather than predicting only whether a given object x belongs to a certain class y, we may also want to take a certain confidence level into account. In this case, $f(x)$ becomes a real-valued function, even though $y \in \{-1, 1\}$.

In this case, $\operatorname{sgn}(f(x))$ denotes the class label, and the absolute value $|f(x)|$ the confidence of the prediction. Corresponding loss functions will depend on the product $yf(x)$ to assess the quality of the estimate. The *soft margin* loss function, as introduced by Bennett and Mangasarian [40, 111], is defined as

Soft Margin Loss

$$c(x, y, f(x)) = \max(0, 1 - yf(x)) = \begin{cases} 0 & \text{if } yf(x) \geq 1, \\ 1 - yf(x) & \text{otherwise.} \end{cases} . \tag{3.3}$$

In some cases [348, 125] (see also Section 10.6.2) the squared version of (3.3) provides an expression that can be minimized more easily;

$$c(x, y, f(x)) = \max(0, 1 - yf(x))^2. \tag{3.4}$$

Logistic Loss

The soft margin loss closely resembles the so-called *logistic* loss function (cf. [251], as well as Problem 3.1 and Section 16.1.1);

$$c(x, y, f(x)) = \ln\left(1 + \exp\left(-yf(x)\right)\right). \tag{3.5}$$

We will derive this loss function in Section 3.3.1. It is used in order to associate a probabilistic meaning with $f(x)$.

Note that in both (3.3) and (3.5) (nearly) no penalty occurs if $yf(x)$ is sufficiently large, i.e. if the patterns are classified correctly with large confidence. In particular, in (3.3) a minimum confidence of 1 is required for zero loss. These loss functions

1. A *false positive* is a point which the classifier erroneously assigns to class 1, a *false negative* is erroneously assigned to class -1.

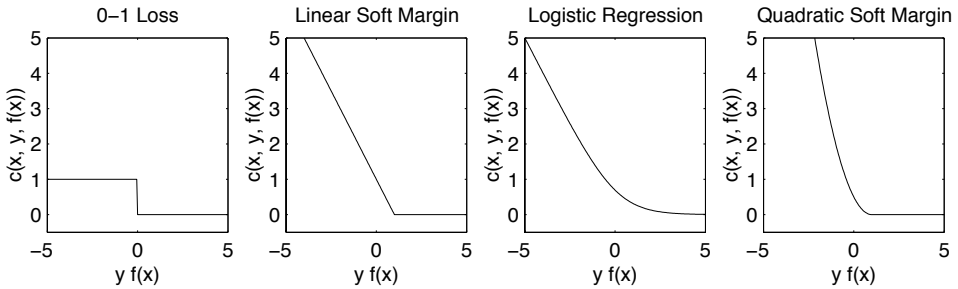


Figure 3.1 From left to right: 0-1 loss, linear soft margin loss, logistic regression, and quadratic soft margin loss. Note that both soft margin loss functions are upper bounds on the 0-1 loss.

led to the development of *large margin classifiers* (see [491, 460, 504] and Chapter 5 for further details). Figure 3.1 depicts various popular loss functions.[2]

Multi Class Discrimination

Matters are more complex when dealing with more than two classes. Each type of misclassification could potentially incur a different loss, leading to an $M \times M$ matrix (M being the number of classes) with positive off-diagonal and zero diagonal entries. It is still a matter of ongoing research in which way a confidence level should be included in such cases (cf. [41, 311, 593, 161, 119]).

3.1.2 Regression

When estimating real-valued quantities, it is usually the size of the difference $y - f(x)$, i.e. the amount of misprediction, rather than the product $yf(x)$, which is used to determine the quality of the estimate. For instance, this can be the actual loss incurred by mispredictions (e.g., the loss incurred by mispredicting the value of a financial instrument at the stock exchange), provided the latter is known and computationally tractable.[3] Assuming location independence, in most cases the loss function will be of the type

$$c(x, y, f(x)) = \tilde{c}(f(x) - y). \tag{3.7}$$

See Figure 3.2 below for several regression loss functions. Below we list the ones most common in kernel methods.

---

2. Other popular loss functions from the generalized linear model context include the inverse complementary log-log function. It is given by

$$c(x, y, f(x)) = 1 - \exp(-\exp(yf(x))). \tag{3.6}$$

This function, unfortunately, is not convex and therefore it will not lead to a convex optimization problem. However, it has nice robustness properties and therefore we think that it should be investigated in the present context.

3. As with classification, computational tractability is one of the primary concerns. This is not always satisfying from a statistician's point of view, yet it is crucial for any practical implementation of an estimation algorithm.

Squared Loss

The popular choice is to minimize the sum of squares of the residuals $f(x) - y$. As we shall see in Section 3.3.1, this corresponds to the assumption that we have additive normal noise corrupting the observations y_i. Consequently we minimize

$$c(x, y, f(x)) = (f(x) - y)^2 \text{ or equivalently } \tilde{c}(\xi) = \xi^2. \tag{3.8}$$

For convenience of subsequent notation, $\frac{1}{2}\xi^2$ rather than ξ^2 is often used.

ε-insensitive Loss and ℓ_1 Loss

An extension of the soft margin loss (3.3) to regression is the *ε-insensitive* loss function [561, 572, 562]. It is obtained by symmetrization of the "hinge" of (3.3),

$$\tilde{c}(\xi) = \max(|\xi| - \varepsilon, 0) =: |\xi|_\varepsilon. \tag{3.9}$$

The idea behind (3.9) is that deviations up to ε should not be penalized, and all further deviations should incur only a linear penalty. Setting $\varepsilon = 0$ leads to an ℓ_1 loss, i.e., to minimization of the sum of absolute deviations. This is written

$$\tilde{c}(\xi) = |\xi|. \tag{3.10}$$

We will study these functions in more detail in Section 3.4.2.

Practical Considerations

For efficient implementations of learning procedures, it is crucial that loss functions satisfy certain properties. In particular, they should be cheap to compute, have a small number of discontinuities (if any) in the first derivative, and be convex in order to ensure the uniqueness of the solution (see Chapter 6 and also Problem 3.6 for details). Moreover, we may want to obtain solutions that are computationally efficient, which may disregard a certain number of training points. This leads to conditions such as vanishing derivatives for a range of function values $f(x)$. Finally, requirements such as outlier resistance are also important for the construction of estimators.

3.2 Test Error and Expected Risk

Now that we have determined how errors should be penalized on specific instances $(x, y, f(x))$, we have to find a method to combine these (local) penalties. This will help us to assess a particular estimate f.

In the following, we will assume that there exists a probability distribution $\mathrm{P}(x, y)$ on $\mathcal{X} \times \mathcal{Y}$ which governs the data generation and underlying functional dependency. Moreover, we denote by $\mathrm{P}(y|x)$ the *conditional* distribution of y given x, and by $d\mathrm{P}(x, y)$ and $d\mathrm{P}(y|x)$ the integrals with respect to the distributions $\mathrm{P}(x, y)$ and $\mathrm{P}(y|x)$ respectively (cf. Section B.1.3).

3.2.1 Exact Quantities

Unless stated otherwise, we assume that the data (x, y) are drawn iid (independent and identically distributed, see Section B.1) from $\mathrm{P}(x, y)$. Whether or not we have

Squared Loss

The popular choice is to minimize the sum of squares of the residuals $f(x) - y$. As we shall see in Section 3.3.1, this corresponds to the assumption that we have additive normal noise corrupting the observations y_i. Consequently we minimize

$$c(x, y, f(x)) = (f(x) - y)^2 \text{ or equivalently } \tilde{c}(\xi) = \xi^2. \tag{3.8}$$

For convenience of subsequent notation, $\frac{1}{2}\xi^2$ rather than ξ^2 is often used.

ε-insensitive Loss and ℓ_1 Loss

An extension of the soft margin loss (3.3) to regression is the *ε-insensitive* loss function [561, 572, 562]. It is obtained by symmetrization of the "hinge" of (3.3),

$$\tilde{c}(\xi) = \max(|\xi| - \varepsilon, 0) =: |\xi|_\varepsilon. \tag{3.9}$$

The idea behind (3.9) is that deviations up to ε should not be penalized, and all further deviations should incur only a linear penalty. Setting $\varepsilon = 0$ leads to an ℓ_1 loss, i.e., to minimization of the sum of absolute deviations. This is written

$$\tilde{c}(\xi) = |\xi|. \tag{3.10}$$

We will study these functions in more detail in Section 3.4.2.

Practical Considerations

For efficient implementations of learning procedures, it is crucial that loss functions satisfy certain properties. In particular, they should be cheap to compute, have a small number of discontinuities (if any) in the first derivative, and be convex in order to ensure the uniqueness of the solution (see Chapter 6 and also Problem 3.6 for details). Moreover, we may want to obtain solutions that are computationally efficient, which may disregard a certain number of training points. This leads to conditions such as vanishing derivatives for a range of function values $f(x)$. Finally, requirements such as outlier resistance are also important for the construction of estimators.

3.2 Test Error and Expected Risk

Now that we have determined how errors should be penalized on specific instances $(x, y, f(x))$, we have to find a method to combine these (local) penalties. This will help us to assess a particular estimate f.

In the following, we will assume that there exists a probability distribution $\mathrm{P}(x, y)$ on $\mathcal{X} \times \mathcal{Y}$ which governs the data generation and underlying functional dependency. Moreover, we denote by $\mathrm{P}(y|x)$ the *conditional* distribution of y given x, and by $d\mathrm{P}(x, y)$ and $d\mathrm{P}(y|x)$ the integrals with respect to the distributions $\mathrm{P}(x, y)$ and $\mathrm{P}(y|x)$ respectively (cf. Section B.1.3).

3.2.1 Exact Quantities

Unless stated otherwise, we assume that the data (x, y) are drawn iid (independent and identically distributed, see Section B.1) from $\mathrm{P}(x, y)$. Whether or not we have

knowledge of the test patterns at training time[4] makes a significant difference in the design of learning algorithms. In the latter case, we will want to minimize the *test error* on that *specific* test set; in the former case, the *expected* error over *all possible* test sets.

Definition 3.2 (Test Error) *Assume that we are not only given the* training *data* $\{x_1, \dots, x_m\}$ *along with target values* $\{y_1, \dots y_m\}$ *but also the* test *patterns* $\{x'_1, \dots x'_{m'}\}$ *on which we would like to predict* y'_i $(i = 1, \dots, m')$*. Since we already know* x'_i*, all we should care about is to minimize the* expected error *on the* test set*. We formalize this in the following definition*

Transduction Problem

$$R_{\text{test}}[f] := \frac{1}{m'} \sum_{i=1}^{m'} \int_{\mathcal{Y}} c(x'_i, y, f(x'_i)) d\mathrm{P}(y|x'_i). \tag{3.11}$$

Unfortunately, this problem, referred to as *transduction*, is quite difficult to address, both computationally and conceptually, see [562, 267, 37, 211]. Instead, one typically considers the case where no knowledge about test patterns is available, as described in the following definition.

Definition 3.3 (Expected Risk) *If we have no knowledge about the test patterns (or decide to ignore them) we should minimize the expected error over all possible training patterns. Hence we have to minimize the expected loss with respect to* P *and c*

$$R[f] := \mathbf{E}\left[R_{\text{test}}[f]\right] = \mathbf{E}\left[c(x, y, f(x))\right] = \int_{\mathcal{X}\times\mathcal{Y}} c(x, y, f(x)) d\mathrm{P}(x, y). \tag{3.12}$$

Here the integration is carried out with respect to the distribution $\mathrm{P}(x, y)$. Again, just as (3.11), this problem is intractable, since we do not know $\mathrm{P}(x, y)$ explicitly. Instead, we are only given the training patterns (x_i, y_i). The latter, however, allow us to replace the unknown distribution $\mathrm{P}(x, y)$ by its empirical estimate.

To study connections between loss functions and density models, it will be convenient to assume that there exists a density $p(x, y)$ corresponding to $\mathrm{P}(x, y)$. This means that we may replace $\int d\mathrm{P}(x, y)$ by $\int p(x, y) dx dy$ and the appropriate measure on $\mathcal{X} \times \mathcal{Y}$. Such a density $p(x, y)$ need not always exist (see Section B.1 for more details) but we will not give further heed to these concerns at present.

3.2.2 Approximations

Unfortunately, this change in notation did not solve the problem. All we have at our disposal is the actual training data. What one usually does is replace $p(x, y)$ by the *empirical density*

Empirical Density

$$p_{\text{emp}}(x, y) := \frac{1}{m} \sum_{i=1}^{m} \delta_{x_i}(x) \delta_{y_i}(y). \tag{3.13}$$

4. The test *outputs*, however, are not available during training.

Here $\delta_{x'}(x)$ denotes the δ-distribution, satisfying $\int \delta_{x'}(x) f(x) dx = f(x')$. The hope is that replacing p by p_{emp} will lead to a quantity that is "reasonably close" to the expected risk. This will be the case if the class of possible solutions f is sufficiently limited [568, 571]. The issue of closeness with regard to different estimators will be discussed in further detail in Chapters 5 and 12. Substituting $p_{\text{emp}}(x, y)$ into (3.12) leads to the empirical risk:

Definition 3.4 (Empirical Risk) *The empirical risk is defined as*

$$R_{\text{emp}}[f] := \int_{\mathcal{X}\times\mathcal{Y}} c(x, y, f(x)) p_{\text{emp}}(x, y) dx dy = \frac{1}{m} \sum_{i=1}^{m} c(x_i, y_i, f(x_i)). \tag{3.14}$$

M-Estimator

This quantity has the advantage that, given the training data, we can readily compute and also minimize it. This constitutes a particular case of what is called an *M-estimator* in statistics. Estimators of this type are studied in detail in the field of empirical processes [554]. As pointed out in Section 3.1, it is crucial to understand that although our particular M-estimator is built from minimizing a loss, this need not always be the case. From a decision-theoretic point of view, the question of which loss to choose is a separate issue, which is dictated by the problem at hand as well as the goal of trying to evaluate the performance of estimation methods, rather than by the problem of trying to define a particular estimation method [582, 166, 43].

Ill-Posed Problems

These considerations aside, it may appear as if (3.14) is the answer to our problems, and all that remains to be done is to find a suitable class of functions $\mathcal{F} \ni f$ such that we can minimize $R_{\text{emp}}[f]$ with respect to $\mathcal{F}$. Unfortunately, determining $\mathcal{F}$ is quite difficult (see Chapters 5 and 12 for details). Moreover, the minimization of $R_{\text{emp}}[f]$ can lead to an ill-posed problem [538, 370]. We will show this with a simple example.

Example of an Ill-Posed Problem

Assume that we want to solve a regression problem using the quadratic loss function (3.8) given by $c(x, y, f(x) = (y - f(x))^2$. Moreover, assume that we are dealing with a linear class of functions,[5] say

$$\mathcal{F} := \left\{ f \,\middle|\, f(x) = \sum_{i=1}^{n} \alpha_i f_i(x) \text{ with } \alpha_i \in \mathbb{R} \right\}, \tag{3.15}$$

where the f_i are functions mapping $\mathcal{X}$ to $\mathbb{R}$.

We want to find the minimizer of R_{emp}, i.e.,

$$\underset{f\in\mathcal{F}}{\text{minimize}}\, R_{\text{emp}}[f] = \underset{\boldsymbol{\alpha}\in\mathbb{R}^n}{\text{minimize}} \frac{1}{m} \sum_{i=1}^{m} \left(y_i - \sum_{j=1}^{n} \alpha_j f_j(x_i) \right)^2. \tag{3.16}$$

5. In the simplest case, assuming $\mathcal{X}$ is contained in a vector space, these could be functions that extract coordinates of x; in other words, $\mathcal{F}$ would be the class of linear functions on $\mathcal{X}$.

Computing the derivative of $R_{\text{emp}}[f]$ with respect to $\boldsymbol{\alpha}$ and defining $F_{ij} := f_i(x_j)$, we can see that the minimum of (3.16) is achieved if

$$F^\top \mathbf{y} = F^\top F \boldsymbol{\alpha}. \tag{3.17}$$

A sufficient condition for (3.17) is $\boldsymbol{\alpha} = \left(F^\top F\right)^{-1} F^\top \mathbf{y}$ where $\left(F^\top F\right)^{-1}$ denotes the (pseudo-)inverse of the matrix.

Condition of a Matrix

If $F^\top F$ has a bad condition number (i.e. the quotient between the largest and the smallest eigenvalue of $F^\top F$ is large), it is numerically difficult [423, 530] to solve (3.17) for $\boldsymbol{\alpha}$. Furthermore, if $n > m$, i.e. if we have more basis functions f_i than training patterns x_i, there will exist a subspace of solutions with dimension at least $n - m$, satisfying (3.17). This is undesirable both practically (speed of computation) and theoretically (we would have to deal with a whole class of solutions rather than a single one).

One might also expect that if $\mathcal{F}$ is too rich, the discrepancy between $R_{\text{emp}}[f]$ and $R[f]$ could be large. For instance, if F is an $m \times m$ matrix of full rank, $\mathcal{F}$ contains an f that predicts all target values y_i correctly on the training data. Nevertheless, we cannot expect that we will also obtain zero prediction error on unseen points. Chapter 4 will show how these problems can be overcome by adding a so-called regularization term to $R_{\text{emp}}[f]$.

3.3 A Statistical Perspective

Given a particular pattern $\tilde{x}$, we may want to ask what risk we can expect for it, and with which *probability* the corresponding loss is going to occur. In other words, instead of (or in addition to) $\mathbf{E}\left[c(\tilde{x}, y, f(\tilde{x})\right]$ for a fixed $\tilde{x}$, we may want to know the distribution of y given $\tilde{x}$, i.e., $\mathrm{P}(y|\tilde{x})$.

(Bayesian) statistics (see [338, 432, 49, 43] and also Chapter 16) often attempt to estimate the density corresponding to the random variables (x, y), and in some cases, we may really *need* information about $p(x, y)$ to arrive at the desired conclusions given the training data (e.g., medical diagnosis). However, we always have to keep in mind that if we model the density p first, and subsequently, based on this approximation, compute a minimizer of the expected risk, we will have to make two approximations. This could lead to inferior or at least not easily predictable results. Therefore, wherever possible, we should avoid solving a more general problem, since additional approximation steps might only make the estimates worse [561].

3.3.1 Maximum Likelihood Estimation

All this said, we still may want to compute the conditional density $p(y|x)$. For this purpose we need to model how y is generated, based on some underlying dependency $f(x)$; thus, we specify the functional form of $p(y|x, f(x))$ and maximize

Computing the derivative of $R_{\text{emp}}[f]$ with respect to $\boldsymbol{\alpha}$ and defining $F_{ij} := f_i(x_j)$, we can see that the minimum of (3.16) is achieved if

$$F^\top \mathbf{y} = F^\top F \boldsymbol{\alpha}. \tag{3.17}$$

A sufficient condition for (3.17) is $\boldsymbol{\alpha} = \left(F^\top F\right)^{-1} F^\top \mathbf{y}$ where $\left(F^\top F\right)^{-1}$ denotes the (pseudo-)inverse of the matrix.

Condition of a Matrix

If $F^\top F$ has a bad condition number (i.e. the quotient between the largest and the smallest eigenvalue of $F^\top F$ is large), it is numerically difficult [423, 530] to solve (3.17) for $\boldsymbol{\alpha}$. Furthermore, if $n > m$, i.e. if we have more basis functions f_i than training patterns x_i, there will exist a subspace of solutions with dimension at least $n - m$, satisfying (3.17). This is undesirable both practically (speed of computation) and theoretically (we would have to deal with a whole class of solutions rather than a single one).

One might also expect that if $\mathcal{F}$ is too rich, the discrepancy between $R_{\text{emp}}[f]$ and $R[f]$ could be large. For instance, if F is an $m \times m$ matrix of full rank, $\mathcal{F}$ contains an f that predicts all target values y_i correctly on the training data. Nevertheless, we cannot expect that we will also obtain zero prediction error on unseen points. Chapter 4 will show how these problems can be overcome by adding a so-called regularization term to $R_{\text{emp}}[f]$.

3.3 A Statistical Perspective

Given a particular pattern $\tilde{x}$, we may want to ask what risk we can expect for it, and with which *probability* the corresponding loss is going to occur. In other words, instead of (or in addition to) $\mathbf{E}\left[c(\tilde{x}, y, f(\tilde{x})\right]$ for a fixed $\tilde{x}$, we may want to know the distribution of y given $\tilde{x}$, i.e., $\mathrm{P}(y|\tilde{x})$.

(Bayesian) statistics (see [338, 432, 49, 43] and also Chapter 16) often attempt to estimate the density corresponding to the random variables (x, y), and in some cases, we may really *need* information about $p(x, y)$ to arrive at the desired conclusions given the training data (e.g., medical diagnosis). However, we always have to keep in mind that if we model the density p first, and subsequently, based on this approximation, compute a minimizer of the expected risk, we will have to make two approximations. This could lead to inferior or at least not easily predictable results. Therefore, wherever possible, we should avoid solving a more general problem, since additional approximation steps might only make the estimates worse [561].

3.3.1 Maximum Likelihood Estimation

All this said, we still may want to compute the conditional density $p(y|x)$. For this purpose we need to model how y is generated, based on some underlying dependency $f(x)$; thus, we specify the functional form of $p(y|x, f(x))$ and maximize

the expression with respect to f. This will provide us with the function f that is *most likely* to have generated the data.

Definition 3.5 (Likelihood) *The likelihood of a sample $(x_1, y_1), \ldots (x_m, y_m)$ given an underlying functional dependency f is given by*

$$p(\{x_1, \ldots, x_m\}, \{y_1, \ldots, y_m\}|f) = \prod_{i=1}^{m} p(x_i, y_i|f) = \prod_{i=1}^{m} p(y_i|x_i, f)p(x_i) \tag{3.18}$$

Strictly speaking the likelihood only depends on the values $f(x_1), \ldots, f(x_m)$ rather than being a functional of f itself. To keep the notation simple, however, we write $p(\{x_1, \ldots, x_m\}, \{y_1, \ldots, y_m\}|f)$ instead of the more heavyweight expression $p(\{x_1, \ldots, x_m\}, \{y_1, \ldots, y_m\}|\{f(x_1), \ldots, f(x_m)\})$.

For practical reasons, we convert products into sums by taking the negative logarithm of $\mathrm{P}(\{x_1, \ldots, x_m\}, \{y_1, \ldots, y_m\}|f)$, an expression which is then conveniently minimized. Furthermore, we may drop the $p(x_i)$ from (3.18), since they do not depend on f. Thus maximization of (3.18) is equivalent to minimization of the

Log-Likelihood

Log-Likelihood

$$\mathfrak{L}[f] := \sum_{i=1}^{m} -\ln p(y_i|x_i, f). \tag{3.19}$$

Regression

Remark 3.6 (Regression Loss Functions) *Minimization of $\mathfrak{L}[f]$ and of $R_{\mathrm{emp}}[f]$ coincide if the loss function c is chosen according to*

$$c(x, y, f(x)) = -\ln p(y|x, f). \tag{3.20}$$

Assuming that the target values y were generated by an underlying functional dependency f plus additive noise ξ with density p_ξ, i.e. $y_i = f_{\mathrm{true}}(x_i) + \xi_i$, we obtain

$$c(x, y, f(x)) = -\ln p_\xi(y - f(x)). \tag{3.21}$$

Things are slightly different in classification. Since all we are interested in is the probability that pattern x has label 1 or -1 (assuming binary classification), we can transform the problem into one of estimating the logarithm of the probability

Classification

that a pattern assumes its correct label.

Remark 3.7 (Classification Loss Functions) *We have a finite set of labels, which allows us to model $\mathrm{P}(y|f(x))$ directly, instead of modelling a density. In the binary classification case (classes 1 and -1) this problem becomes particularly easy, since all we have to do is assume functional dependency underlying $\mathrm{P}(1|f(x))$: this immediately gives us $\mathrm{P}(-1|f(x)) = 1 - \mathrm{P}(1|f(x))$. The link to loss functions is established via*

$$c(x, y, f(x)) = -\ln \mathrm{P}(y|f(x)). \tag{3.22}$$

The same result can be obtained by minimizing the cross entropy[6] between the classifica-

6. In the case of discrete variables the cross entropy between two distributions P and Q is defined as $\sum_i \mathrm{P}(i) \ln \mathrm{Q}(i)$.

Table 3.1 Common loss functions and corresponding density models according to Remark 3.6. As a shorthand we use $\tilde{c}(f(x) - y) := c(x, y, f(x))$.

	loss function $\tilde{c}(\xi)$	density model $p(\xi)$
ε-insensitive	$\lvert\xi\rvert_\varepsilon$	$\frac{1}{2(1+\varepsilon)} \exp(-\lvert\xi\rvert_\varepsilon)$
Laplacian	$\lvert\xi\rvert$	$\frac{1}{2} \exp(-\lvert\xi\rvert)$
Gaussian	$\frac{1}{2}\xi^2$	$\frac{1}{\sqrt{2\pi}} \exp(-\frac{\xi^2}{2})$
Huber's robust loss	$\begin{cases} \frac{1}{2\sigma}(\xi)^2 & \text{if } \lvert\xi\rvert \le \sigma \\ \lvert\xi\rvert - \frac{\sigma}{2} & \text{otherwise} \end{cases}$	$\propto \begin{cases} \exp(-\frac{\xi^2}{2\sigma}) & \text{if } \lvert\xi\rvert \le \sigma \\ \exp(\frac{\sigma}{2} - \lvert\xi\rvert) & \text{otherwise} \end{cases}$
Polynomial	$\frac{1}{d}\lvert\xi\rvert^d$	$\frac{d}{2\Gamma(1/d)} \exp(-\lvert\xi\rvert^d)$
Piecewise polynomial	$\begin{cases} \frac{1}{d\sigma^{d-1}}\lvert\xi\rvert^d & \text{if } \lvert\xi\rvert \le \sigma \\ \lvert\xi\rvert - \sigma\frac{d-1}{d} & \text{otherwise} \end{cases}$	$\propto \begin{cases} \exp(-\frac{\lvert\xi\rvert^d}{d\sigma^{d-1}}) & \text{if } \lvert\xi\rvert\sigma \\ \exp(\sigma\frac{d-1}{d} - \lvert\xi\rvert) & \text{otherwise} \end{cases}$

tion labels y_i and the probabilities $p(y|f(x))$, as is typically done in a generalized linear models context (see e.g., [355, 232, 163]). For binary classification (with $y \in \{\pm 1\}$) we obtain

$$c(x, y, f(x)) = \frac{1+y}{2} \ln \mathrm{P}(y = 1|f(x)) + \frac{1-y}{2} \ln \mathrm{P}(y = -1|f(x)). \tag{3.23}$$

When substituting the actual values for y into (3.23), this reduces to (3.22).

At this point we have a choice in modelling $\mathrm{P}(y = 1|f(x))$ to suit our needs. Possible models include the logistic transfer function, the probit model, the inverse complementary log-log model. See Section 16.3.5 for a more detailed discussion of the choice of such *link functions*. Below we explain connections in some more detail for the logistic link function.

For a logistic model, where $\mathrm{P}(y = \pm 1|x, f) \propto \exp(\pm\frac{1}{2}f(x))$, we obtain after normalization

$$\mathrm{P}(y = 1|x, f) := \frac{\exp(f(x))}{1 + \exp(f(x))} \tag{3.24}$$

and consequently $-\ln \mathrm{P}(y = 1|x, f) = \ln(1 + \exp(-f(x)))$. We thus recover (3.5) as the loss function for classification. Choices other than (3.24) for a map $\mathbb{R} \to [0, 1]$ will lead to further loss functions for classification. See [579, 179, 596] and Section 16.1.1 for more details on this subject.

It is important to note that not every loss function used in classification corresponds to such a density model (recall that in this case, the probabilities have to add up to 1 for any value of $f(x)$). In fact, one of the most popular loss functions, the soft margin loss (3.3), does not enjoy this property. A discussion of these issues can be found in [521].

Examples

Table 3.1 summarizes common loss functions and the corresponding density models as defined by (3.21), some of which were already presented in Section 3.1. It is an exhaustive list of the loss functions that will be used in this book for regression. Figure 3.2 contains graphs of the functions.

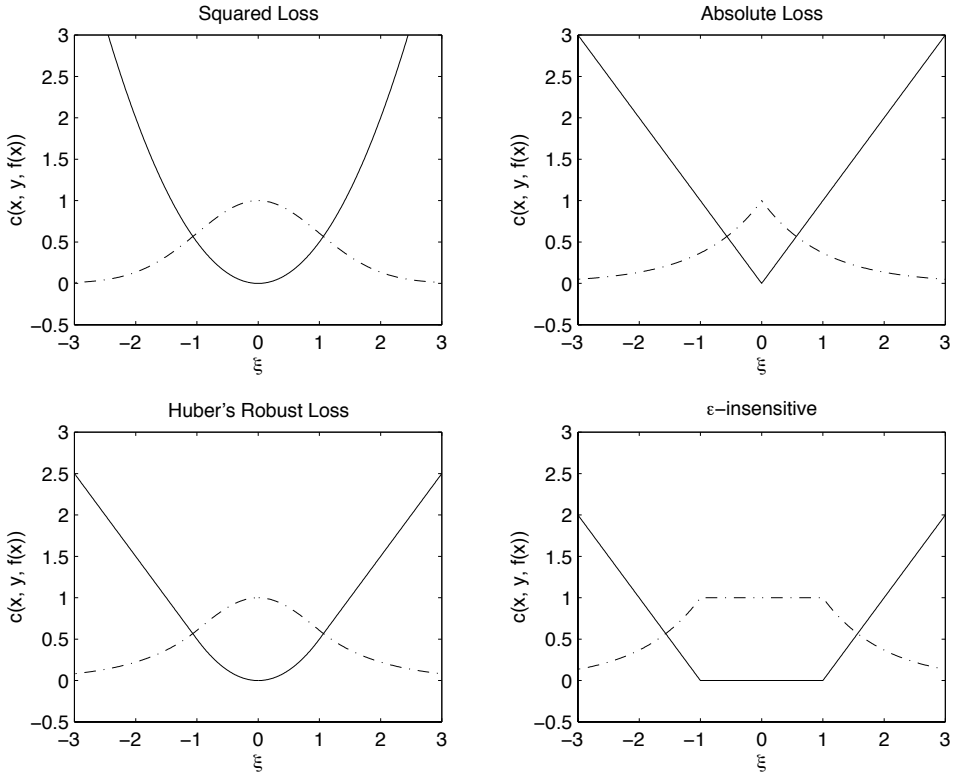


Figure 3.2 Graphs of loss functions and corresponding density models. upper left: Gaussian, upper right: Laplacian, lower left: Huber's robust, lower right: ε-insensitive.

Practical Considerations

We conclude with a few cautionary remarks. The loss function resulting from a maximum likelihood reasoning might be non-convex. This might spell trouble when we try to find an efficient solution of the corresponding minimization problem. Moreover, we made a very strong assumption by claiming to know $\mathrm{P}(y|x, f)$ explicitly, which was necessary in order to evaluate (3.20).

Finally, the solution we obtain by minimizing the log-likelihood depends on the class of functions $\mathcal{F}$. So we are in no better situation than by minimizing $R_{\mathrm{emp}}[f]$, albeit with the additional constraint, that the loss functions $c(x, y, f(x))$ must correspond to a probability density.

3.3.2 Efficiency

The above reasoning could mislead us into thinking that the choice of loss function is rather arbitrary, and that there exists no good means of assessing the performance of an estimator. In the present section we will develop tools which can be used to compare estimators that are derived from different loss functions. For this purpose we need to introduce additional statistical concepts which deal with the efficiency of an estimator. Roughly speaking, these give an indication of how

"noisy" an estimator is with respect to a reference estimator.

We begin by formalizing the concept of an estimator. Denote by $\mathrm{P}(y|\theta)$ a distribution of y depending (amongst other variables) on the parameters θ, and by $Y = \{y_1, \ldots, y_m\}$ an m-sample drawn iid from $\mathrm{P}(y|\theta)$. Note that the use of the symbol y bears no relation to the y_i that are outputs of some functional dependency (cf. Chapter 1). We employ this symbol because some of the results to be derived will later be applied to the outputs of SV regression.

Estimator

Next, we introduce the *estimator* $\hat{\theta}(Y)$ of the parameters θ, based on Y. For instance, $\mathrm{P}(y|\theta)$ could be a Gaussian with fixed variance and mean θ, and $\hat{\theta}(Y)$ could be the estimator $(1/m)\sum_{i=1}^{m} y_i$.

To avoid cumbersome notation, we use the shorthand

$$\mathbf{E}_\theta\left[\xi(y)\right] := \mathbf{E}_{\mathrm{P}(y|\theta)}\left[\xi(y)\right] = \int \xi(y) d\mathrm{P}(y|\theta), \tag{3.25}$$

to express expectations of a random variable $\xi(y)$ with respect to $\mathrm{P}(y|\theta)$. One criterion that we might impose on an estimator is that it be unbiased, i.e., that on average, it tells us the correct value of the parameter it attempts to estimate.

Definition 3.8 (Unbiased Estimator) *An unbiased estimator $\hat{\theta}(Y)$ of the parameters θ in $\mathrm{P}(y|\theta)$ satisfies*

$$\mathbf{E}_\theta\left[\hat{\theta}(Y)\right] = \theta. \tag{3.26}$$

In this section, we will focus on unbiased estimators. In general, however, the estimators we are dealing with in this book will not be unbiased. In fact, they will have a bias towards 'simple', low-complexity functions. Properties of such estimators are more difficult to deal with, which is why, for the sake of simplicity, we restrict ourselves to the unbiased case in this section. Note, however, that "biasedness" is not a bad property by itself. On the contrary, there exist cases as the one described by James and Stein [262] where biased estimators consistently outperform unbiased estimators in the finite sample size setting, both in terms of variance and prediction error.

A possible way to compare unbiased estimators is to compute their variance. Other quantities such as moments of higher order or maximum deviation properties would be valid criteria as well, yet for historical and practical reasons the variance has become a standard tool to benchmark estimators. The Fisher information matrix is crucial for this purpose since it will tell us via the Cramér-Rao bound (Theorem 3.11) the minimal possible variance for an unbiased estimator. The idea is that the smaller the variance, the lower (typically) the probability that $\hat{\theta}(Y)$ will deviate from θ by a large amount. Therefore, we can use the variance as a possible one number summary to compare different estimators.

Definition 3.9 (Score Function, Fisher Information, Covariance) *Assume there exists a density $p(y|\theta)$ for the distribution $\mathrm{P}(y|\theta)$ such that $\ln p(y|\theta)$ is differentiable with*

Score Function

respect to θ. The score $V_\theta(Y)$ *of* $\mathrm{P}(y|\theta)$ *is a random variable defined by*[7]

$$V_\theta(Y) := \partial_\theta \ln p(Y|\theta) = \partial_\theta \sum_{i=1}^{m} \ln p(y_i|\theta) = \sum_{i=1}^{m} \frac{\partial_\theta p(y_i|\theta)}{p(y_i|\theta)}. \tag{3.27}$$

This score tells us how much the likelihood of the data depends on the different components of θ, and thus, in the maximum likelihood procedure, how much the data affect the choice of θ. The covariance of $V_\theta(Y)$ is called the Fisher information matrix *I. It is given by*

Fisher Information

$$I_{ij} := \mathbf{E}_\theta \left[\partial_{\theta_i} \ln p(Y|\theta) \cdot \partial_{\theta_j} \ln p(Y|\theta) \right]. \tag{3.28}$$

Covariance

and the covariance matrix *B of the estimator $\hat{\theta}(Y)$ is defined by*

$$B_{ij} := \mathbf{E}_\theta \left[\left(\hat{\theta}_i - \mathbf{E}_\theta \left[\hat{\theta}_i \right] \right) \left(\hat{\theta}_j - \mathbf{E}_\theta \left[\hat{\theta}_j \right] \right) \right]. \tag{3.29}$$

The covariance matrix B tells us the amount of variation of the estimator. It can therefore be used (e.g., by Chebychev's inequality) to bound the probability that $\hat{\theta}(Y)$ deviates from θ by more than a certain amount.

Remark 3.10 (Expected Value of Fisher Score) *One can check that the expected value of $V_\theta(Y)$ is* 0 *since*

$$\mathbf{E}_\theta [V_\theta(Y)] = \int p(Y|\theta) \partial_\theta \ln p(Y|\theta) dY = \partial_\theta \int p(Y|\theta) dY = \partial_\theta 1 = 0. \tag{3.30}$$

Average Fisher Score Vanishes

In other words, the contribution of Y to the adjustment of θ averages to 0 *over all possible Y, drawn according to* $\mathrm{P}(Y|\theta)$*. Equivalently we could say that the average likelihood for Y drawn according to* $\mathrm{P}(Y|\theta)$ *is extremal, provided we choose θ: the derivative of the expected likelihood of the data* $\mathbf{E}_\theta \left[\ln \mathrm{P}(Y|\theta) \right]$ *with respect to θ vanishes. This is also what we expect, namely that the "proper" distribution is on average the one with the highest likelihood.*

The following theorem gives a lower bound on the variance of an estimator, i.e. B is found in terms of the Fisher information I. This is useful to determine how well a given estimator performs with respect to the one with the lowest possible variance.

Theorem 3.11 (Cramér and Rao [425]) *Any unbiased estimator $\hat{\theta}(Y)$ satisfies*

$$\det IB \geq 1. \tag{3.31}$$

Proof We prove (3.31) for the scalar case. The extension to matrices is left as an exercise (see Problem 3.10). Using the Cauchy-Schwarz inequality, we obtain

$$\left(\mathbf{E}_\theta \left[(V_\theta(Y) - \mathbf{E}_\theta [V_\theta(Y)]) \left(\hat{\theta}(Y) - \mathbf{E}_\theta \left[\hat{\theta}(Y) \right] \right) \right] \right)^2 \tag{3.32}$$

$$\leq \mathbf{E}_\theta \left[(V_\theta(Y) - \mathbf{E}_\theta [V_\theta(Y)])^2 \right] \mathbf{E}_\theta \left[\left(\hat{\theta}(Y) - \mathbf{E}_\theta \left[\hat{\theta}(Y) \right] \right)^2 \right] = IB. \tag{3.33}$$

7. Recall that $\partial_\theta p(Y|\theta)$ is the gradient of $p(Y|\theta)$ with respect to the parameters $\theta_1, \ldots, \theta_n$.

At the same time, $\mathbf{E}_\theta[V_\theta(Y)] = 0$ implies that

$$\left(\mathbf{E}_\theta\left[(V_\theta(Y) - \mathbf{E}_\theta[V_\theta(Y)])\left(\hat{\theta}(Y) - \mathbf{E}_\theta\left[\hat{\theta}(Y)\right]\right)\right]\right)^2 \tag{3.34}$$

$$= \mathbf{E}_\theta\left[V_\theta(Y)\hat{\theta}(Y)\right]^2 \tag{3.35}$$

$$= \left(\int p(Y|\theta)V_\theta(Y)\hat{\theta}(Y)dY\right)^2$$

$$= \left(\partial_\theta \int p(Y|\theta)\hat{\theta}(Y)dY\right)^2 = (\partial_\theta\theta)^2 = 1, \tag{3.36}$$

since we may interchange integration by Y and ∂_θ. ■

Eq. (3.31) lends itself to the definition of a one-number summary of the properties of an estimator, namely how closely the inequality is met.

Definition 3.12 (Efficiency) *The statistical efficiency e of an estimator $\hat{\theta}(Y)$ is defined as*

$$e := 1/\det IB. \tag{3.37}$$

The closer e is to 1, the lower the variance of the corresponding estimator $\hat{\theta}(Y)$. For a special class of estimators minimizing loss functions, the following theorem allows us to compute B and e efficiently.

Asymptotic Variance

Theorem 3.13 (Murata, Yoshizawa, Amari [379, Lemma 3]) *Assume that $\hat{\theta}$ is defined by $\hat{\theta}(Y) := \operatorname{argmin}_\theta d(Y, \theta)$ and that d is a twice differentiable function in θ. Then asymptotically, for increasing sample size $m \to \infty$, the variance B is given by $B = Q^{-1}GQ^{-1}$. Here*

$$G_{ij} := \operatorname{cov}_\theta\left[\partial_{\theta_i} d(Y, \theta), \partial_{\theta_j} d(Y, \theta)\right] \text{ and} \tag{3.38}$$

$$Q_{ij} := \mathbf{E}_\theta\left[\partial^2_{\theta_i\theta_j} d(Y, \theta)\right], \tag{3.39}$$

and therefore $e = (\det Q)^2/(\det IG)$.

This means that for the class of estimators defined via d, the evaluation of their asymptotic efficiency can be conveniently achieved via (3.38) and (3.39). For scalar valued estimators $\theta(Y) \in \mathbb{R}$, these expressions can be greatly simplified to

$$I = \int \left(\partial_\theta \ln p(Y|\theta)\right)^2 d\mathrm{P}(Y|\theta), \tag{3.40}$$

$$G = \int \left(\partial_\theta d(Y, \theta)\right)^2 d\mathrm{P}(Y|\theta), \tag{3.41}$$

$$Q = \int \partial^2_\theta d(Y, \theta) d\mathrm{P}(Y|\theta). \tag{3.42}$$

Finally, in the case of continuous densities, Theorem 3.13 may be extended to piecewise twice differentiable continuous functions d, by convolving the latter with a twice differentiable smoothing kernel, and letting the width of the smoothing kernel converge to zero. We will make use of this observation in the next section when studying the efficiency of some estimators.

The current section concludes with the proof that the maximum likelihood estimator meets the Cramér-Rao bound.

Theorem 3.14 (Efficiency of Maximum Likelihood [118, 218, 43]) *The maximum likelihood estimator (cf. (3.18) and (3.19)) given by*

$$\hat{\theta}(Y) := \operatorname*{argmax}_{\theta} \ln p(Y|\theta) = \operatorname*{argmin}_{\theta} \mathcal{L}[\theta] \tag{3.43}$$

is asymptotically efficient ($e = 1$).

To keep things simple we will prove (3.43) only for the class of twice differentiable continuous densities by applying Theorem 3.13. For a more general proof see [118, 218, 43].

Proof By construction, G is equal to the Fisher information matrix, if we choose d according to (3.43). Hence a sufficient condition is that $Q = -I$, which is what we show below. To this end we expand the integrand of (3.42),

$$\partial_\theta^2 d(Y,\theta) = \partial_\theta^2 \ln p(Y|\theta) = \frac{\partial_\theta^2 p(Y|\theta)}{p(Y|\theta)} - \left(\frac{\partial_\theta p(Y|\theta)}{p(Y|\theta)}\right)^2 = \frac{\partial_\theta^2 p(Y|\theta)}{p(Y|\theta)} - V_\theta^2(Y). \tag{3.44}$$

The expectation of the second term in (3.44) equals $-I$. We now show that the expectation of the first term vanishes;

$$\int p(Y|\theta)\frac{\partial_\theta^2 p(Y|\theta)}{p(Y|\theta)}dY = \partial_\theta^2 \int p(Y|\theta)dY = \partial_\theta^2 1 = 0. \tag{3.45}$$

Hence $Q = -I$ and thus $e = Q^2/(IG) = 1$. This proves that the maximum likelihood estimator is asymptotically efficient. ∎

It appears as if the best thing we could do is to use the maximum likelihood (ML) estimator. Unfortunately, reality is not quite as simple as that. First, the above statement holds only asymptotically. This leads to the (justified) suspicion that for finite sample sizes we may be able to do better than ML estimation. Second, practical considerations such as the additional goal of sparse decomposition may lead to the choice of a non-optimal loss function.

Finally, we may not know the true density model, which is required for the definition of the maximum likelihood estimator. We can try to make an educated guess; bad guesses of the class of densities, however, can lead to large errors in the estimation (see, e.g., [251]). This prompted the development of robust estimators.

3.4 Robust Estimators

So far, in order to make any practical predictions, we had to *assume* a certain class of distributions from which P(Y) was chosen. Likewise, in the case of risk functionals, we also assumed that training and test data are identically distributed. This section provides tools to safeguard ourselves against cases where the above

The current section concludes with the proof that the maximum likelihood estimator meets the Cramér-Rao bound.

Theorem 3.14 (Efficiency of Maximum Likelihood [118, 218, 43]) *The maximum likelihood estimator (cf. (3.18) and (3.19)) given by*

$$\hat{\theta}(Y) := \operatorname*{argmax}_{\theta} \ln p(Y|\theta) = \operatorname*{argmin}_{\theta} \mathcal{L}[\theta] \tag{3.43}$$

is asymptotically efficient ($e = 1$).

To keep things simple we will prove (3.43) only for the class of twice differentiable continuous densities by applying Theorem 3.13. For a more general proof see [118, 218, 43].

Proof By construction, G is equal to the Fisher information matrix, if we choose d according to (3.43). Hence a sufficient condition is that $Q = -I$, which is what we show below. To this end we expand the integrand of (3.42),

$$\partial_\theta^2 d(Y,\theta) = \partial_\theta^2 \ln p(Y|\theta) = \frac{\partial_\theta^2 p(Y|\theta)}{p(Y|\theta)} - \left(\frac{\partial_\theta p(Y|\theta)}{p(Y|\theta)}\right)^2 = \frac{\partial_\theta^2 p(Y|\theta)}{p(Y|\theta)} - V_\theta^2(Y). \tag{3.44}$$

The expectation of the second term in (3.44) equals $-I$. We now show that the expectation of the first term vanishes;

$$\int p(Y|\theta)\frac{\partial_\theta^2 p(Y|\theta)}{p(Y|\theta)}dY = \partial_\theta^2 \int p(Y|\theta)dY = \partial_\theta^2 1 = 0. \tag{3.45}$$

Hence $Q = -I$ and thus $e = Q^2/(IG) = 1$. This proves that the maximum likelihood estimator is asymptotically efficient. ∎

It appears as if the best thing we could do is to use the maximum likelihood (ML) estimator. Unfortunately, reality is not quite as simple as that. First, the above statement holds only asymptotically. This leads to the (justified) suspicion that for finite sample sizes we may be able to do better than ML estimation. Second, practical considerations such as the additional goal of sparse decomposition may lead to the choice of a non-optimal loss function.

Finally, we may not know the true density model, which is required for the definition of the maximum likelihood estimator. We can try to make an educated guess; bad guesses of the class of densities, however, can lead to large errors in the estimation (see, e.g., [251]). This prompted the development of robust estimators.

3.4 Robust Estimators

So far, in order to make any practical predictions, we had to *assume* a certain class of distributions from which P(Y) was chosen. Likewise, in the case of risk functionals, we also assumed that training and test data are identically distributed. This section provides tools to safeguard ourselves against cases where the above

assumptions are not satisfied.

Outliers

More specifically, we would like to avoid a certain fraction ν of 'bad' observations (often also referred to as 'outliers') seriously affecting the quality of the estimate. This implies that the influence of individual patterns should be bounded from above. Huber [250] gives a detailed list of desirable properties of a robust estimator. We refrain from reproducing this list at present, or committing to a particular definition of robustness.

As usual for the estimation of location parameter context (i.e. estimation of the expected value of a random variable) we assume a specific parametric form of $p(Y|\theta)$, namely

$$p(Y|\theta) = \prod_{i=1}^{m} p(y_i|\theta) = \prod_{i=1}^{m} p(y_i - \theta). \tag{3.46}$$

Unless stated otherwise, this is the formulation we will use throughout this section.

3.4.1 Robustness via Loss Functions

Huber's idea [250] in constructing a robust estimator was to take a loss function as provided by the maximum likelihood framework, and modify it in such a way as to limit the influence of each individual pattern. This is done by providing an upper bound on the slope of $-\ln p(Y|\theta)$. We shall see that methods such as the trimmed mean or the median are special cases thereof. The ε-insensitive loss function can also be viewed as a trimmed estimator. This will lead to the development of adaptive loss functions in the subsequent sections. We begin with the main theorem of this section.

Mixture Densities

Theorem 3.15 (Robust Loss Functions (Huber [250])) *Let $\mathfrak{P}$ be a class of densities formed by*

$$\mathfrak{P} := \{p|p = (1-\varepsilon)p_0 + \varepsilon p_1\} \text{ where } \varepsilon \in (0,1) \text{ and } p_0 \text{ are known}. \tag{3.47}$$

Moreover assume that both p_0 and p_1 are symmetric with respect to the origin, their logarithms are twice continuously differentiable, $\ln p_0$ is convex and known, and p_1 is unknown. Then the density

$$\bar{p}(\theta) := (1-\varepsilon) \begin{cases} p_0(\theta) & \text{if } |\theta| \leq \theta_0 \\ p_0(\theta_0)e^{-k(|\theta|-\theta_0)} & \text{otherwise} \end{cases} \tag{3.48}$$

is robust in the sense that the maximum likelihood estimator corresponding to (3.48) has minimum variance with respect to the "worst" possible density $p_{\text{worst}} = (1-\varepsilon)p_0 + \varepsilon p_1$: it is a saddle point (located at p_{worst}) in terms of variance with respect to the true density $p \in \mathfrak{P}$ and the density $\bar{p} \in \mathfrak{P}$ used in estimating the location parameter. This means that no density p has larger variance than p_{worst} and that for $p = p_{\text{worst}}$ no estimator is better than the one where $\bar{p} = p_{\text{worst}}$, as used in the robust estimator.

The constants $k > 0$ and θ_0 are obtained by the normalization condition, that $\bar{p}$ be a

proper density and that the first derivative in $\ln \bar{p}$ *be continuous.*

Proof To show that $\bar{p}$ is a saddle point in $\mathfrak{P}$ we have to prove that (a) no estimation procedure other than the one using $\ln \bar{p}$ as the loss function has lower variance for the density $\bar{p}$, and that (b) no density has higher variance than $\bar{p}$ if $\ln \bar{p}$ is used as loss function. Part (a) follows immediately from the Cramér-Rao theorem (Th. 3.11); part (b) can be proved as follows.

We use Theorem 3.13, and a proof technique pointed out in [559], to compute the variance of an estimator using $\ln \bar{p}$ as loss function;

$$B = \frac{\int \left(\partial_\theta \ln \bar{p}(y|\theta)\right)^2 \left((1-\varepsilon)p_0(y|\theta) + \varepsilon p'(y|\theta)\right) dy}{\int \partial_\theta^2 \ln \bar{p}(y|\theta) \left((1-\varepsilon)p_0(y|\theta) + \varepsilon p'(y|\theta)\right) dy}. \tag{3.49}$$

Here p' is an arbitrary density which we will choose such that B is maximized. By construction,

$$\left(\partial_\theta \ln \bar{p}(y|\theta)\right)^2 = \begin{cases} \left(\partial_\theta \ln p_0(y|\theta)\right)^2 \leq k^2 & \text{if } |y-\theta| \leq \theta_0, \\ k^2 & \text{otherwise,} \end{cases} \tag{3.50}$$

$$\partial_\theta^2 \ln \bar{p}(y|\theta) = \begin{cases} \partial_\theta^2 \ln p_0(y|\theta) \geq 0 & \text{if } |y-\theta| \leq \theta_0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.51}$$

Thus any density p' which is 0 in $[-\theta_0, \theta_0]$ will minimize the denominator (the term depending on p' will be 0, which is the lowest obtainable value due to (3.51)), and maximize the numerator, since in the latter the contribution of p' is always limited to $k^2\varepsilon$. Now $\varepsilon^{-1}\left(\bar{p} - (1-\varepsilon)p_0\right)$ is exactly such a density. Hence the saddle point property holds. ■

Remark 3.16 (Robustness Classes) *If we have more knowledge about the class of densities* $\mathfrak{P}$*, a different loss function will have the saddle point property. For instance, using a similar argument as above, one can show that the normal distribution is robust in the class of all distributions with bounded variance. This implies that among all possible distributions with bounded variance, the estimator of the mean of a normal distribution has the highest variance.*

Likewise, the Laplacian distribution is robust in the class of all symmetric distributions with density $p(0) \geq c$ *for some fixed* $c > 0$ *(see [559, 251] for more details).*

Hence, even though a loss function defined according to Theorem 3.15 is generally desirable, we may be less cautious, and use a different loss function for improved performance, when we have additional knowledge of the distribution.

Remark 3.17 (Mean and Median) *Assume we are dealing with a mixture of a normal distribution with variance* σ^2 *and an additional unknown distribution with weight at most* ε*. It is easy to check that the application of Theorem 3.15 to normal distributions yields Huber's robust loss function from Table 3.1.*

The maximizer of the likelihood (see also Problem 3.17) is a trimmed mean estimator which discards ε *of the data: effectively all* θ_i *deviating from the mean by more than* σ *are*

ignored and the mean is computed from the remaining data. Hence Theorem 3.15 gives a formal justification for this popular type of estimator.

If we let $\varepsilon \to 1$ we recover the median estimator which stems from a Laplacian distribution. Here, all patterns but the median one are discarded.

Trimmed Interval Estimator

Besides the classical examples of loss functions and density models, we might also consider a slightly unconventional estimation procedure: use the average between the k-smallest and the k-largest of all observations θ observations as the estimated mean of the underlying distribution (for sorted observations θ_i with $\theta_i \leq \theta_j$ for $1 \leq i \leq j \leq m$ the estimator computes $(\theta_k + \theta_{m-k+1})/2$). This procedure makes sense, for instance, when we are trying to infer the mean of a random variable generated by roundoff noise (i.e., noise whose density is constant within some bounded interval) plus an additional unknown amount of noise.

Support Patterns

Note that both the patterns strictly *inside* or *outside* an interval of size $[-\varepsilon, \varepsilon]$ around the estimate have no direct influence on the outcome. Only patterns *on* the boundary matter. This is a very similar situation to the behavior of Support Vector Machines in regression, and one can show that it corresponds to the minimizer of the ε-insensitive loss function (3.9). We will study the properties of the latter in more detail in the following section and thereafter show how it can be transformed into an adaptive risk functional.

3.4.2 Efficiency and the ε-Insensitive Loss Function

The tools of Section 3.3.2 allow us to analyze the ε-insensitive loss function in more detail. Even though the asymptotic estimation of a location parameter setting is a gross oversimplification of what is happening in a SV regression estimator (where we estimate a nonparametric function, and moreover have only a limited number of observations at our disposition), it will provide us with useful insights into this more complex case [510, 481].

In a first step, we compute the efficiency of an estimator, for several noise models and amounts of variance, using a density corresponding to the ε-insensitive loss function (cf. Table 3.1);

$$p_\varepsilon(y|\theta) = \frac{1}{2+2\varepsilon}\exp(-|y-\theta|_\varepsilon) = \frac{1}{2+2\varepsilon}\begin{cases} 1 & \text{if } |y-\theta| \leq \varepsilon, \\ \exp(\varepsilon - |y-\theta|) & \text{otherwise.} \end{cases} \tag{3.52}$$

For this purpose we have to evaluate the quantities G (3.41) and Q (3.42) of Theorem 3.13. We obtain

$$G = m\int \left(\partial_\theta \ln p(y|\theta)\right)^2 d\mathrm{P}(y|\theta) = m\left(1 - \int_{-\varepsilon}^{\varepsilon} p(y|\theta)dy\right), \tag{3.53}$$

$$Q = m\int \partial_\theta^2 \ln p(y|\theta) d\mathrm{P}(y|\theta) = m\left(p(-\varepsilon+\theta|\theta) + p(\varepsilon+\theta|\theta)\right). \tag{3.54}$$

The Fisher information I of m iid random variables distributed according to p_θ is m-times the value of a single random variable. Thus all dependencies on m in e cancel out and we can limit ourselves to the case of $m = 1$ for the analysis of the

efficiency of estimators.

Now we may check what happens if we use the ε-insensitive loss function for different types of noise model. For the sake of simplicity we begin with Gaussian noise.

Example 3.18 (Gaussian Noise) *Assume that y is normally distributed with zero mean (i.e. $\theta = 0$) and variance σ. By construction, the minimum obtainable variance is $I^{-1} = \sigma^2$ (recall that $m = 1$). Moreover (3.53) and (3.54) yield*

$$\frac{G}{Q^2} = \sigma^2 \exp\left(\frac{\varepsilon^2}{\sigma^2}\right)\left(1 - \operatorname{erf}\frac{\varepsilon}{\sqrt{2}\sigma}\right). \tag{3.55}$$

The efficiency $e = \frac{Q^2}{GI}$ is maximized for $\varepsilon = 0.6120\sigma$. This means that if the underlying noise model is Gaussian with variance σ and *we have to use an ε-insensitive loss function to estimate a location parameter, the most efficient estimator from this family is given by $\varepsilon = 0.6120\sigma$.*

The consequence of (3.55) is that the optimal value of ε scales linearly with σ. Of course, we could just use squared loss in such a situation, but in general, we will not know the exact noise model, and squared loss does not lead to robust estimators. The following lemma (which will come handy in the next section) shows that this is a general property of the ε-insensitive loss.

Lemma 3.19 (Linear Dependency between ε-Tube Width and Variance) *Denote by p a symmetric density with variance $\sigma > 0$. Then the optimal value of ε (i.e. the value that achieves maximum asymptotic efficiency) for an estimator using the ε-insensitive loss is given by*

$$\varepsilon_{\text{opt}} = \sigma \operatorname*{argmin}_{\tau} \frac{1}{\left(p_{\text{std}}(-\tau) + p_{\text{std}}(\tau)\right)^2}\left(1 - \int_{-\tau}^{\tau} p_{\text{std}}(\tau')d\tau'\right), \tag{3.56}$$

where $p_{\text{std}}(\tau) := \sigma p(\sigma\tau + \theta|\theta)$ is the standardized version of $p(y|\theta)$, i.e. it is obtained by rescaling $p(y|\theta)$ to zero mean and unit variance.

Since p_{std} is independent of σ, we have a linear dependency between ε_{opt} and σ. The scaling factor depends on the noise model.

Proof We prove (3.56) by rewriting the efficiency $e(\varepsilon)$ in terms of p_{std} via $p(y|\theta) = \sigma^{-1} p_{\text{std}}(\sigma^{-1}(y - \theta))$. This yields

$$e(\varepsilon) = \frac{Q^2}{IG} = \frac{\left(\sigma^{-1}p_{\text{std}}(-\sigma^{-1}\varepsilon) + \sigma^{-1}p_{\text{std}}(\sigma^{-1}\varepsilon)\right)^2}{\sigma^{-2}\left(1 - \int_{-\varepsilon}^{\varepsilon}\sigma^{-1}p_{\text{std}}(\sigma^{-1}\theta)d\theta\right)} = \frac{\left(p_{\text{std}}(-\sigma^{-1}\varepsilon) + p_{\text{std}}(\sigma^{-1}\varepsilon)\right)^2}{\left(1 - \int_{-\sigma^{-1}\varepsilon}^{\sigma^{-1}\varepsilon} p_{\text{std}}(\theta)d\theta\right)}$$

The maximum of $e(\varepsilon)$ does not depend directly on ε, but on $\sigma^{-1}\varepsilon$ (which is independent of σ). Hence we can find $\operatorname{argmax}_{\varepsilon} e(\varepsilon)$ by solving (3.56). ■

Lemma 3.19 made it apparent that in order to adjust ε we have to know σ beforehand. Unfortunately, the latter is usually unknown at the beginning of the

estimation procedure.[8] The solution to this dilemma is to make ε adaptive.

3.4.3 Adaptive Loss Functions

We again consider the trimmed mean estimator, which discards a predefined fraction of largest and smallest samples. This method belongs to the more general class of quantile estimators, which base their estimates on the value of samples in a certain quantile. The latter methods do not require prior knowledge of the variance, and adapt to whatever scale is required. What we need is a technique which connects σ (in Huber's robust loss function) or ε (in the ε-insensitive loss case) with the deviations between the estimate $\hat{\theta}$ and the random variables y_i.

Let us analyze what happens to the negative log likelihood, if, in the ε-insensitive case, we change ε to $\varepsilon + \delta$ (with $\delta \in \mathbb{R}$) while keeping $\hat{\theta}$ fixed. In particular we assume that $|\delta|$ is chosen sufficiently small such that for all $i = 1, \ldots, m$,

$$|\hat{\theta} - y_i| \begin{cases} \leq \varepsilon + \delta & \text{if } |\hat{\theta} - y_i| < \varepsilon \\ \geq \varepsilon + \delta & \text{if } |\hat{\theta} - y_i| > \varepsilon \end{cases} \tag{3.57}$$

Moreover denote by $m_<, m_=, m_>$ the number of samples for which $|\hat{\theta} - y_i|$ is less than, equal to, or greater than ε, respectively. Then

$$\begin{aligned} \sum_{i=1}^{m} |\hat{\theta} - y_i|_{\varepsilon+\delta} &= \sum_{|\hat{\theta}-y_i|<\varepsilon} |\hat{\theta} - y_i|_\varepsilon + \sum_{|\hat{\theta}-y_i|>\varepsilon} |\hat{\theta} - y_i|_\varepsilon - m_> \delta + \sum_{|\hat{\theta}-y_i|=\varepsilon} |\hat{\theta} - y_i|_{\varepsilon+\delta} \\ &= \sum_{i=1}^{m} |\hat{\theta} - y_i|_\varepsilon - \begin{cases} m_> \delta & \text{if } \delta > 0, \\ (m_< + m_=)\delta & \text{otherwise.} \end{cases} \end{aligned} \tag{3.58}$$

In other words, the amount by which the loss changes depends only on the quantiles at ε. What happens if we make ε itself a variable of the optimization problem? By the scaling properties of (3.58) one can see that for $\nu \in [0, 1]$

$$\underset{\hat{\theta},\varepsilon}{\text{minimize}} \frac{1}{m} \sum_{i=1}^{m} |\hat{\theta} - y_i|_\varepsilon - \nu\varepsilon \tag{3.59}$$

ν-Property

is minimized if ε is chosen such that

$$\frac{m_>}{m} \leq \nu \leq \frac{m_> + m_=}{m}. \tag{3.60}$$

This relation holds since at the solution $(\hat{\theta}, \varepsilon)$ the solution also has to be optimal wrt. ε alone while keeping $\hat{\theta}$ fixed. In the latter case, however, the derivatives of

8. The obvious question is why one would ever like to choose an ε-insensitive loss in the presence of Gaussian noise in the first place. If the complexity of the function expansion is of no concern and the highest accuracy is required, squared loss is to be preferred. In most cases, however, it is not quite clear what *exactly* the type of the additive noise model is. This is when we would like to have a more conservative estimator. In practice, the ε-insensitive loss has been shown to work rather well on a variety of tasks (Chapter 9).

the log-likelihood (i.e. error) term wrt. ε at the solution are given by $\frac{m_>}{m}$ and $\frac{m_> + m_=}{m}$ on the left and right hand side respectively.[9] These have to cancel with ν which proves the claim. Furthermore, computing the derivative of (3.59) with respect to $\hat{\theta}$ shows that the number of samples outside the interval $[\theta - \varepsilon, \theta + \varepsilon]$ has to be equal on both halves $(-\infty, \theta - \varepsilon)$ and $(\theta + \varepsilon, \infty)$. We have the following theorem:

Theorem 3.20 (Quantile Estimation as Optimization Problem [481]) *A quantile procedure to estimate the mean of a distribution by taking the average of the samples at the $\frac{\nu}{2}$th and $(1 - \frac{\nu}{2})$th quantile is equivalent to minimizing (3.59). In particular,*

1. *ν is an upper bound on the fraction of samples outside the interval $[\theta - \varepsilon, \theta + \varepsilon]$.*
2. *ν is a lower bound on the fraction of samples outside the interval $]\theta - \varepsilon, \theta + \varepsilon[$.*
3. *If the distribution $p(\theta)$ is continuous, for all $\nu \in [0, 1]$*

$$\lim_{m \to \infty} \mathrm{P}\left\{ \frac{m_=}{m} < \varepsilon \right\} = 1 \text{ for all } \varepsilon > 0. \tag{3.61}$$

One might question the practical advantage of this method over direct trimming of the sample Y. In fact, the use of (3.59) is not recommended if all we want is to estimate θ. That said, (3.59) does allow us to employ trimmed estimation in the nonparametric case, cf. Chapter 9.

Extension to General Robust Estimators

Unfortunately, we were unable to find a similar method for Huber's robust loss function, since in this case the change in the negative log-likelihood incurred by changing σ not only involves the (statistical) rank of y_i, but also the exact location of samples with $|y_i - \theta| < \sigma$.

One way to overcome this problem is re-estimate σ adaptively while minimizing a term similar to (3.59) (see [180] for details in the context of boosting, Section 10.6.3 for a discussion of online estimation techniques, or [251] for a general overview).

3.4.4 Optimal Choice of ν

Let us return to the ε-insensitive loss. A combination of Theorems 3.20, 3.13 and Lemma 3.19 allows us to compute optimal values of ν for various distributions, provided that an ε-insensitive loss function is to be used in the estimation procedure.[10]

The idea is to determine the optimal value of ε for a fixed density $p(y|\theta)$ via (3.56), and compute the corresponding fraction ν of patterns outside the interval $[-\varepsilon + \theta, \varepsilon + \theta]$.

9. Strictly speaking, the derivative is not defined at ε; the lhs and rhs values are defined, however, which is sufficient for our purpose.
10. This is not optimal in the sense of Theorem 3.15, which suggests the use of a more adapted loss function. However (as already stated in the introduction of this chapter), algorithmic or technical reasons such as computationally efficient solutions or limited memory may provide sufficient motivation to use such a loss function.

Table 3.2 Optimal ν and ε for various degrees of polynomial additive noise.

Polynomial Degree d	1	2	3	4	5
Optimal ν	1	0.5405	0.2909	0.1898	0.1384
Optimal ε for unit variance	0	0.6120	1.1180	1.3583	1.4844
Polynomial Degree d	6	7	8	9	10
Optimal ν	0.1080	0.0881	0.0743	0.0641	0.0563
Optimal ε for unit variance	1.5576	1.6035	1.6339	1.6551	1.6704

Theorem 3.21 (Optimal Choice of ν) *Denote by p a symmetric density with variance $\sigma > 0$ and by p_{std} the corresponding rescaled density with zero mean and unit variance. Then the optimal value of ν (i.e. the value that achieves maximum asymptotic efficiency) for an estimator using the ε-insensitive loss is given by*

$$\nu = 1 - \int_{-\varepsilon}^{\varepsilon} p_{\text{std}}(y)dy \tag{3.62}$$

where ε is chosen according to (3.56). This expression is independent of σ.

Proof The independence of σ follows from the fact that ν depends only on p_{std}. Next we show (3.62). For a given density p, the asymptotically optimal value of ε is given by Lemma 3.19. The average fraction of patterns outside the interval $[\hat{\theta} - \varepsilon_{\text{opt}}, \hat{\theta} + \varepsilon_{\text{opt}}]$ is

$$\nu = 1 - \int_{-\varepsilon_{\text{opt}}+\theta}^{\varepsilon_{\text{opt}}+\theta} p(y|\theta)dy = 1 - \int_{-\sigma^{-1}\varepsilon_{\text{opt}}}^{\sigma^{-1}\varepsilon_{\text{opt}}} p_{\text{std}}(y)dy, \tag{3.63}$$

which depends only on $\sigma^{-1}\varepsilon_{\text{opt}}$ and is thus independent of σ. Combining (3.63) with (3.56) yields the theorem. ∎

This means that given the *type* of additive noise, we *can* determine the value of ν such that it yields the asymptotically most efficient estimator *independent* of the *level* of the noise. These theoretical predictions have since been confirmed rather accurately in a set of regression experiments [95].

Let us now look at some special cases.

Example 3.22 (Optimal ν for Polynomial Noise) *Arbitrary polynomial noise models ($\propto e^{-|\theta|^d}$) with unit variance can be written as*

$$p(y) = c_p \exp\left(-c'_p|y|^p\right) \text{ where } c_p = \frac{1}{2}\sqrt{\frac{\Gamma\left(3/d\right)}{\Gamma\left(1/d\right)}}\frac{d}{\Gamma\left(1/d\right)} \text{ and } c'_p = \left(\sqrt{\frac{\Gamma\left(3/d\right)}{\Gamma\left(1/d\right)}}\right)^d,$$

where $\Gamma(x)$ is the gamma function. Figure 3.3 shows ν_{opt} for polynomial degrees in the interval $[1, 10]$. For convenience, the explicit numerical values are repeated in Table 3.2.

Heavy Tails → Large ν

Observe that as the distribution becomes "lighter-tailed", the optimal ν decreases; in other words, we may then use a larger amount of the data for the purpose of estimation. This is reasonable since it is only for very long tails of the distribution (data with many

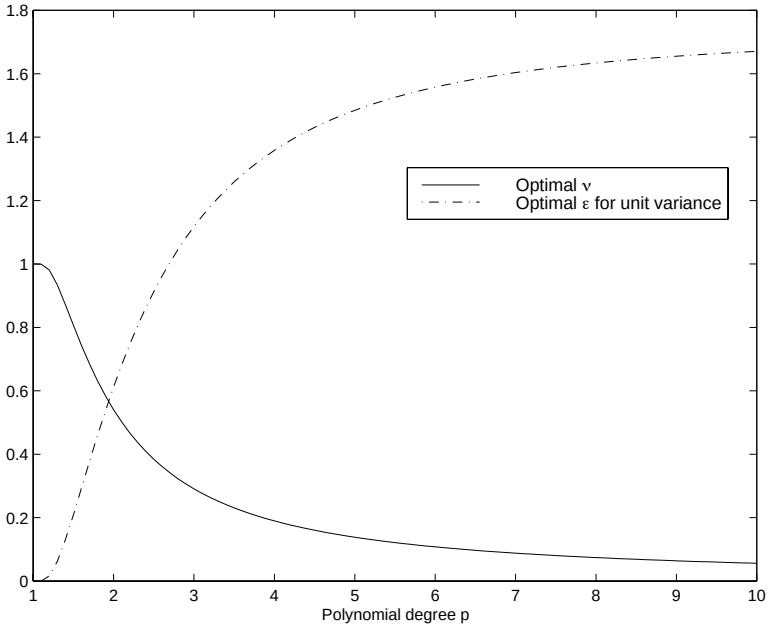


Figure 3.3 Optimal ν and ε for various degrees of polynomial additive noise.

outliers) that we have to be conservative and discard a large fraction of observations.

Even though we derived these relations solely for the case where a single number (θ) has to be estimated, experiments show that the same scaling properties hold for the nonparametric case. It is still an open research problem to establish this connection exactly.

As we shall see, in the nonparametric case, the effect of ν will be that it both determines the number of Support Vectors (i.e., the number of basis functions needed to expand the solution) and also the fraction of function values $f(x_i)$ with deviation larger than ε from the corresponding observations. Further information on this topic, both from the statistical and the algorithmic point of view, can be found in Section 9.3.

3.5 Summary

We saw in this chapter that there exist two complementary concepts as to how risk and loss functions should be designed. The first one is data driven and uses the incurred loss as its principal guideline, possibly modified in order to suit the need of numerical efficiency. This leads to loss functions and the definitions of empirical and expected risk.

A second method is based on the idea of estimating (or at least approximating) the distribution which may be responsible for generating the data. We showed

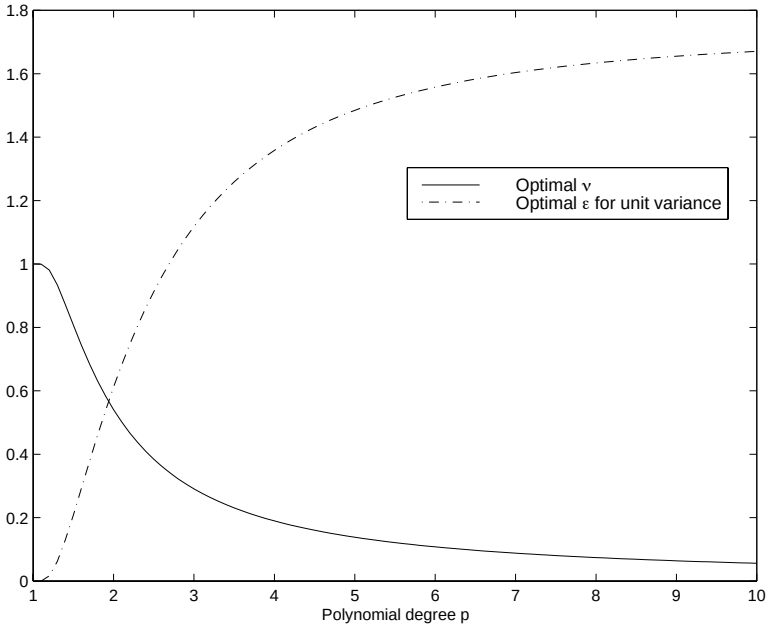


Figure 3.3 Optimal ν and ε for various degrees of polynomial additive noise.

outliers) that we have to be conservative and discard a large fraction of observations.

Even though we derived these relations solely for the case where a single number (θ) has to be estimated, experiments show that the same scaling properties hold for the nonparametric case. It is still an open research problem to establish this connection exactly.

As we shall see, in the nonparametric case, the effect of ν will be that it both determines the number of Support Vectors (i.e., the number of basis functions needed to expand the solution) and also the fraction of function values $f(x_i)$ with deviation larger than ε from the corresponding observations. Further information on this topic, both from the statistical and the algorithmic point of view, can be found in Section 9.3.

3.5 Summary

We saw in this chapter that there exist two complementary concepts as to how risk and loss functions should be designed. The first one is data driven and uses the incurred loss as its principal guideline, possibly modified in order to suit the need of numerical efficiency. This leads to loss functions and the definitions of empirical and expected risk.

A second method is based on the idea of estimating (or at least approximating) the distribution which may be responsible for generating the data. We showed

that in a Maximum Likelihood setting this concept is rather similar to the notions of risk and loss, with $c(x, y, f(x)) = -\ln p(y|x, f(x))$ as the link between both quantities.

This point of view allowed us to analyze the properties of estimators in more detail and provide lower bounds on the performance of unbiased estimators, i.e. the Cramér-Rao theorem. The latter was then used as a benchmarking tool for various loss functions and density models, such as the ε-insensitive loss. The consequence of this analysis is a corroboration of experimental findings that there exists a linear correlation between the amount of noise in the observations and the optimal width of ε.

This, in turn, allowed us to construct adaptive loss functions which adjust themselves to the amount of noise, much like trimmed mean estimators. These formulations can be used directly in mathematical programs, leading to ν-SV algorithms in subsequent chapters. The question of which choices are optimal in a finite sample size setting remains an open research problem.

3.6 Problems

3.1 (Soft Margin and Logistic Regression •) *The soft margin loss function c_{soft} and the logistic loss c_{logist} are asymptotically almost the same; show that*

$$\lim_{f\to\infty} \big(c_{\text{soft}}(x, 1, f) - c_{\text{logist}}(x, 1, f)\big) = 1 \tag{3.64}$$

$$\lim_{f\to-\infty} \big(c_{\text{soft}}(x, 1, f) - c_{\text{logist}}(x, 1, f)\big) = 0. \tag{3.65}$$

3.2 (Multi-class Discrimination ••) *Assume you have to solve a classification problem with M different classes. Discuss how the number of functions used to solve this task affects the quality of the solution.*

- *How would the loss function look if you were to use only one real-valued function $f : \mathcal{X} \to \mathbb{R}$. Which symmetries are violated in this case (hint: what happens if you permute the classes)?*
- *How many functions do you need if each of them makes a binary decision $f : \mathcal{X} \to \{0, 1\}$?*
- *How many functions do you need in order to make the solution permutation symmetric with respect to the class labels?*
- *How should you assess the classification error? Is it a good idea to use the misclassification rate of one individual function as a performance criterion (hint: correlation of errors)? By how much can this error differ from the total misclassification error?*

3.3 (Mean and Median •) *Assume 8 people want to gather for a meeting; 5 of them live in Stuttgart and 3 in Munich. Where should they meet if (a) they want the total distance traveled by all people to be minimal, (b) they want the average distance traveled per person to be minimal, or (c) they want the average squared distance to be minimal? What happens*

that in a Maximum Likelihood setting this concept is rather similar to the notions of risk and loss, with $c(x, y, f(x)) = -\ln p(y|x, f(x))$ as the link between both quantities.

This point of view allowed us to analyze the properties of estimators in more detail and provide lower bounds on the performance of unbiased estimators, i.e. the Cramér-Rao theorem. The latter was then used as a benchmarking tool for various loss functions and density models, such as the ε-insensitive loss. The consequence of this analysis is a corroboration of experimental findings that there exists a linear correlation between the amount of noise in the observations and the optimal width of ε.

This, in turn, allowed us to construct adaptive loss functions which adjust themselves to the amount of noise, much like trimmed mean estimators. These formulations can be used directly in mathematical programs, leading to ν-SV algorithms in subsequent chapters. The question of which choices are optimal in a finite sample size setting remains an open research problem.

3.6 Problems

3.1 (Soft Margin and Logistic Regression •) *The soft margin loss function c_{soft} and the logistic loss c_{logist} are asymptotically almost the same; show that*

$$\lim_{f\to\infty} \big(c_{\text{soft}}(x, 1, f) - c_{\text{logist}}(x, 1, f)\big) = 1 \tag{3.64}$$

$$\lim_{f\to-\infty} \big(c_{\text{soft}}(x, 1, f) - c_{\text{logist}}(x, 1, f)\big) = 0. \tag{3.65}$$

3.2 (Multi-class Discrimination ••) *Assume you have to solve a classification problem with M different classes. Discuss how the number of functions used to solve this task affects the quality of the solution.*

- *How would the loss function look if you were to use only one real-valued function $f : \mathcal{X} \to \mathbb{R}$. Which symmetries are violated in this case (hint: what happens if you permute the classes)?*
- *How many functions do you need if each of them makes a binary decision $f : \mathcal{X} \to \{0, 1\}$?*
- *How many functions do you need in order to make the solution permutation symmetric with respect to the class labels?*
- *How should you assess the classification error? Is it a good idea to use the misclassification rate of one individual function as a performance criterion (hint: correlation of errors)? By how much can this error differ from the total misclassification error?*

3.3 (Mean and Median •) *Assume 8 people want to gather for a meeting; 5 of them live in Stuttgart and 3 in Munich. Where should they meet if (a) they want the total distance traveled by all people to be minimal, (b) they want the average distance traveled per person to be minimal, or (c) they want the average squared distance to be minimal? What happens*

to the meeting points if one of the 3 *people moves from Munich to Sydney?*

3.4 (Locally Adaptive Loss Functions •••) *Assume that the loss function* $c(x, y, f(x))$ *varies with* x*. What does this mean for the expected loss? Can you give a bound on the latter even if you know* $p(y|x)$ *and* f *at every point but know* c *only on a finite sample (hint: construct a counterexample)? How will things change if* c *cannot vary much with* x*?*

3.5 (Transduction Error •••) *Assume that we want to minimize the test error of misclassification* $R_{\text{test}}[f]$*, given a training sample* $\{(x_1, y_1), \ldots, (x_m, y_m)\}$*, a test sample* $\{x'_1, \ldots, x'_{m'}\}$ *and a loss function* $c(x, y, f(x))$*.*

Show that any loss function $c'(x', f(x'))$ *on the test sample has to be symmetric in* f*, i.e.* $c'(x', f(x')) = c'(x', -f(x'))$*. Prove that no non-constant convex function can satisfy this property. What does this mean for the practical solution of optimization problem? See [267, 37, 211, 103] for details.*

3.6 (Convexity and Uniqueness ••) *Show that the problem of estimating a location parameter (a single scalar) has an interval* $[a, b] \subset \mathbb{R}$ *of equivalent global minima if the loss functions are convex. For non-convex loss functions construct an example where this is not the case.*

3.7 (Linearly Dependent Parameters ••) *Show that in a linear model* $f = \sum_i \alpha_i f_i$ *on* $\mathcal{X}$ *it is impossible to find a unique set of optimal parameters* α_i *if the functions* f_i *are not linearly independent. Does this have any effect on* f *itself?*

3.8 (Ill-posed Problems •••) *Assume you want to solve the problem* $Ax = y$ *where* A *is a symmetric positive definite matrix, i.e., a matrix with nonnegative eigenvalues. If you change* y *to* y'*, how much will the solution* x' *of* $Ax' = y'$ *differ from* x'*. Give lower and upper bounds on this quantity. Hint: decompose* y *into the eigensystem of* A*.*

3.9 (Fisher Map [258] ••) *Show that the map*

$$U_\theta(x) := I^{-\frac{1}{2}} \partial_\theta \ln p(x|\theta) \tag{3.66}$$

maps x *into vectors with zero mean and unit variance. Chapter 13 will use this map to design kernels.*

3.10 (Cramér-Rao Inequality for Multivariate Estimators ••) *Prove equation (3.31). Hint: start by applying the Cauchy-Schwarz inequality to*

$$\left(\det E_{\bar{\theta}}[(\hat{\theta}(\theta) - E_{\bar{\theta}}\hat{\theta}(\theta))(T_\theta(\theta) - E_{\bar{\theta}} T_\theta(\theta))^\top]\right) \tag{3.67}$$

to obtain I *and* B *and compute the expected value coefficient-wise.*

3.11 (Soft Margin Loss and Conditional Probabilities [521] •••) *What is the conditional probability* $p(y|x)$ *corresponding to the soft margin loss function* $c(x, y, f(x)) = \max(0, 1 - yf(x))$*?*

- *How can you fix the problem that the probabilities $p(-1|x)$ and $p(1|x)$ have to sum up to 1?*
- *How does the introduction of a third class ("don't know") change the problem? What is the problem with this approach? Hint: What is the behavior for large $|f(x)|$?*

3.12 (Label Noise ••) *Denote by $\mathrm{P}(y = 1|f(x))$ and $\mathrm{P}(y = -1|f(x))$ the conditional probabilities of labels ± 1 for a classifier output $f(x)$. How will P change if we randomly flip labels with $\eta \in (0, 1)$ probability? How should you adapt your density model?*

3.13 (Unbiased Estimators ••) *Prove that the least mean square estimator is unbiased for arbitrary symmetric distributions. Can you extend the result to arbitrary symmetric losses?*

3.14 (Efficiency of Huber's Robust Estimator ••) *Compute the efficiency of Huber's Robust Estimator in the presence of pure Gaussian noise with unit variance.*

3.15 (Influence and Robustness •••) *Prove that for robust estimators using (3.48) as their density model, the maximum change in the minimizer of the empirical risk is bounded by $\frac{\delta k}{m}$ if a sample θ_i is changed to $\theta_i + \delta$. What happens in the case of Gaussian density models (i.e., squared loss)?*

3.16 (Robustness of Gaussian Distributions [559] •••) *Prove that the normal distribution with variance σ^2 is robust among the class of distributions with bounded variance (by σ^2). Hint: show that we have a saddle point analogous to Theorem 3.15 by exploiting Theorems 3.13 and Theorem 3.14.*

3.17 (Trimmed Mean ••) *Show that under the assumption of an unknown distribution contributing at most ε, Huber's robust loss function for normal distributions leads to a trimmed mean estimator which discards ε of the data.*

3.18 (Optimal ν for Gaussian Noise •) *Give an explicit solution for the optimal ν in the case of additive Gaussian noise.*

3.19 (Optimal ν for Discrete Distribution ••) *Assume that we have a noise model with a discrete distribution of θ, where $\mathrm{P}(\theta = \epsilon) = \mathrm{P}(\theta = -\epsilon) = p_1$, $\mathrm{P}(\theta = 2\epsilon) = \mathrm{P}(\theta = -2\epsilon) = p_2$, $2(p_1 + p_2) = 1$, and $p_1, p_2 \geq 0$. Compute the optimal value of ν.*

5 Elements of Statistical Learning Theory

We now give a more complete exposition of the ideas of statistical learning theory, which we briefly touched on in Chapter 1. We mentioned previously that in order to learn from a small training set, we should try to *explain* the data with a model of *small* capacity; we have not yet justified *why* this is the case, however. This is the main goal of the present chapter.

Overview

We start by revisiting the difference between risk minimization and empirical risk minimization, and illustrating some common pitfalls in machine learning, such as overfitting and training on the test set (Section 5.1). We explain that the motivation for empirical risk minimization is the law of large numbers, but that the classical version of this law is not sufficient for our purposes (Section 5.2). Thus, we need to introduce the statistical notion of *consistency* (Section 5.3). It turns out that consistency of learning algorithms amounts to a law of large numbers, which holds uniformly over all functions that the learning machine can implement (Section 5.4). This crucial insight, due to Vapnik and Chervonenkis, focuses our attention on the set of attainable functions; this set must be restricted in order to have any hope of succeeding. Section 5.5 states probabilistic bounds on the risk of learning machines, and summarizes different ways of characterizing precisely how the set of functions can be restricted. This leads to the notion of *capacity concepts*, which gives us the main ingredients of the typical generalization error bound of statistical learning theory. We do not indulge in a complete treatment; rather, we try to give the main insights to provide the reader with some intuition as to how the different pieces of the puzzle fit together. We end with a section showing an example application of risk bounds for model selection (Section 5.6).

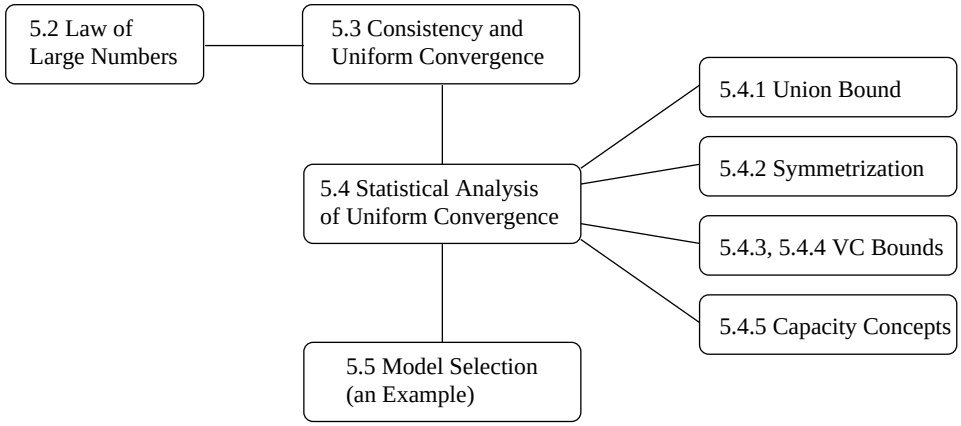
Prerequisites

The chapter attempts to present the material in a fairly non-technical manner, providing intuition wherever possible. Given the nature of the subject matter, however, a limited amount of mathematical background is required. The reader who is not familiar with basic probability theory should first read Section B.1.

5.1 Introduction

Let us start with an example. We consider a regression estimation problem. Suppose we are given empirical observations,

$$(x_1, y_1), \ldots, (x_m, y_m) \in \mathcal{X} \times \mathcal{Y}, \tag{5.1}$$



Regression Example

where for simplicity we take $\mathcal{X} = \mathcal{Y} = \mathbb{R}$. Figure 5.1 shows a plot of such a dataset, along with two possible functional dependencies that could underlie the data. The dashed line represents a fairly complex model, and fits the training data perfectly. The straight line, on the other hand, does not completely "explain" the data, in the sense that there are some residual errors; it is much "simpler," however. A physicist measuring these data points would argue that it cannot be by chance that the measurements almost lie on a straight line, and would much prefer to attribute the residuals to measurement error than to an erroneous model. But is it possible to *characterize* the way in which the straight line is simpler, and why this should imply that it is, in some sense, closer to an underlying true dependency?

Bias-Variance Dilemma

In one form or another, this issue has long occupied the minds of researchers studying the problem of learning. In classical statistics, it has been studied as the *bias-variance dilemma*. If we computed a linear fit for every data set that we ever encountered, then every functional dependency we would ever "discover" would be linear. But this would not come from the data; it would be a *bias* imposed by us. If, on the other hand, we fitted a polynomial of sufficiently high degree to any given data set, we would always be able to fit the data perfectly, but the exact model we came up with would be subject to large fluctuations, depending on

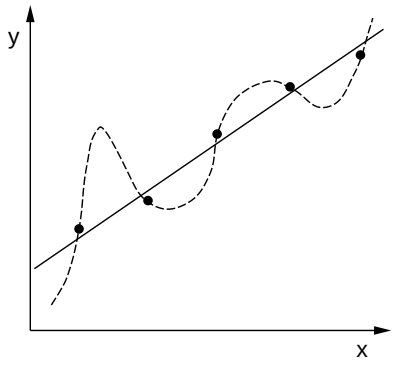


Figure 5.1 Suppose we want to estimate a functional dependence from a set of examples (black dots). Which model is preferable? The complex model perfectly fits all data points, whereas the straight line exhibits residual errors. Statistical learning theory formalizes the role of the *complexity* of the model class, and gives probabilistic guarantees for the validity of the inferred model.

how accurate our measurements were in the first place — the model would suffer from a large *variance*. A related dichotomy is the one between *estimation error* and *approximation error*. If we use a small class of functions, then even the best possible solution will poorly approximate the "true" dependency, while a large class of functions will lead to a large statistical estimation error.

Overfitting

In the terminology of applied machine learning and the design of neural networks, the complex explanation shows *overfitting*, while an overly simple explanation imposed by the learning machine design would lead to *underfitting*. A great deal of research has gone into clever engineering tricks and heuristics; these are used, for instance, to aid in the design of neural networks which will not overfit on a given data set [397]. In neural networks, overfitting can be avoided in a number of ways, such as by choosing a number of hidden units that is not too large, by *stopping* the training procedure early in order not to enforce a perfect explanation of the training set, or by using *weight decay* to limit the size of the weights, and thus of the function class implemented by the network.

Statistical learning theory provides a solid mathematical framework for studying these questions in depth. As mentioned in Chapters 1 and 3, it makes the assumption that the data are generated by sampling from an unknown underlying distribution $P(x, y)$. The learning problem then consists in minimizing the *risk* (or *expected loss* on the test data, see Definition 3.3),

Risk

$$R[f] = \int_{\mathcal{X}\times\mathcal{Y}} c(x, y, f(x))\, dP(x, y). \tag{5.2}$$

Here, c is a loss function. In the case of pattern recognition, where $\mathcal{Y} = \{\pm 1\}$, a common choice is the misclassification error, $c(x, y, f(x)) = \frac{1}{2}|f(x) - y|$.

The difficulty of the task stems from the fact that we are trying to minimize a quantity that we cannot actually evaluate: since we do not know P, we cannot compute the integral (5.2). What we *do* know, however, are the training data (5.1), which are sampled from P. We can thus try to infer a function f from the training sample that is, in some sense, *close* to the one minimizing (5.2). To this end, we need what is called an *induction principle*.

Empirical Risk

One way to proceed is to use the training sample to approximate the integral in (5.2) by a finite sum (see (B.18)). This leads to the empirical risk (Definition 3.4),

$$R_{\text{emp}}[f] = \frac{1}{m}\sum_{i=1}^{m} c(x_i, y_i, f(x_i)), \tag{5.3}$$

and the *empirical risk minimization (ERM) induction principle*, which recommends that we choose an f that minimizes (5.3).

Cast in these terms, the fundamental trade-off in learning can be stated as follows: if we allow f to be taken from a very large class of functions $\mathcal{F}$, we can always find an f that leads to a rather small value of (5.3). For instance, if we allow the use of *all* functions f mapping $\mathcal{X} \to \mathcal{Y}$ (in compact notation, $\mathcal{F} = \mathcal{Y}^{\mathcal{X}}$), then we can minimize (5.3) yet still be distant from the minimizer of (5.2). Considering a

pattern recognition problem, we could set

$$f(x) = \begin{cases} y_i & \text{if } x = x_i \text{ for some } i = 1, \ldots, m \\ 1 & \text{otherwise.} \end{cases} \tag{5.4}$$

This does not amount to any form of learning, however: suppose we are now given a test point drawn from the same distribution, $(x, y) \sim P(x, y)$. If $\mathcal{X}$ is a continuous domain, and we are not in a degenerate situation, the new pattern x will almost never be exactly equal to any of the training inputs x_i. Therefore, the learning machine will almost always predict that $y = 1$. *If we allow* all *functions from $\mathcal{X}$ to $\mathcal{Y}$, then the values of the function at points $x_1, \ldots, x_m$ carry no information about the values at other points.* In this situation, a learning machine cannot do better than chance. This insight lies at the core of the so-called *No-Free-Lunch Theorem* popularized in [608]; see also [254, 48].

The message is clear: if we make no restrictions on the class of functions from which we choose our estimate f, we cannot hope to learn anything. Consequently, machine learning research has studied various ways to implement such restrictions. In statistical learning theory, these restrictions are enforced by taking into account the *complexity* or *capacity* (measured by VC dimension, covering numbers, entropy numbers, or other concepts) of the class of functions that the learning machine can implement.[1]

In the Bayesian approach, a similar effect is achieved by placing *prior distributions $P(f)$* over the class of functions (Chapter 16). This may sound fundamentally different, but it leads to algorithms which are closely related; and on the theoretical side, recent progress has highlighted intriguing connections [92, 91, 353, 238].

5.2 The Law of Large Numbers

Let us step back and try to look at the problem from a slightly different angle. Consider the case of pattern recognition using the misclassification loss function. Given a fixed function f, then for each example, the loss $\xi_i := \frac{1}{2}|f(x_i) - y_i|$ is either

1. As an aside, note that the same problem applies to *training on the test set* (sometimes called *data snooping*): sometimes, people optimize tuning parameters of a learning machine by looking at how they change the results on an independent test set. Unfortunately, once one has adjusted the parameter in this way, the test set is not independent anymore. This is identical to the corresponding problem in training on the *training* set: once we have chosen the function to minimize the training error, the latter no longer provides an unbiased estimate of the test error. Overfitting occurs much faster on the training set, however, than it does on the test set. This is usually due to the fact that the number of tuning parameters of a learning machine is much smaller than the total number of parameters, and thus the capacity tends to be smaller. For instance, an SVM for pattern recognition typically has two tuning parameters, and optimizes m weight parameters (for a training set size of m). See also Problem 5.3 and [461].

pattern recognition problem, we could set

$$f(x) = \begin{cases} y_i & \text{if } x = x_i \text{ for some } i = 1, \ldots, m \\ 1 & \text{otherwise.} \end{cases} \tag{5.4}$$

This does not amount to any form of learning, however: suppose we are now given a test point drawn from the same distribution, $(x, y) \sim P(x, y)$. If $\mathcal{X}$ is a continuous domain, and we are not in a degenerate situation, the new pattern x will almost never be exactly equal to any of the training inputs x_i. Therefore, the learning machine will almost always predict that $y = 1$. *If we allow* all *functions from $\mathcal{X}$ to $\mathcal{Y}$, then the values of the function at points $x_1, \ldots, x_m$ carry no information about the values at other points.* In this situation, a learning machine cannot do better than chance. This insight lies at the core of the so-called *No-Free-Lunch Theorem* popularized in [608]; see also [254, 48].

The message is clear: if we make no restrictions on the class of functions from which we choose our estimate f, we cannot hope to learn anything. Consequently, machine learning research has studied various ways to implement such restrictions. In statistical learning theory, these restrictions are enforced by taking into account the *complexity* or *capacity* (measured by VC dimension, covering numbers, entropy numbers, or other concepts) of the class of functions that the learning machine can implement.[1]

In the Bayesian approach, a similar effect is achieved by placing *prior distributions $P(f)$* over the class of functions (Chapter 16). This may sound fundamentally different, but it leads to algorithms which are closely related; and on the theoretical side, recent progress has highlighted intriguing connections [92, 91, 353, 238].

5.2 The Law of Large Numbers

Let us step back and try to look at the problem from a slightly different angle. Consider the case of pattern recognition using the misclassification loss function. Given a fixed function f, then for each example, the loss $\xi_i := \frac{1}{2}|f(x_i) - y_i|$ is either

1. As an aside, note that the same problem applies to *training on the test set* (sometimes called *data snooping*): sometimes, people optimize tuning parameters of a learning machine by looking at how they change the results on an independent test set. Unfortunately, once one has adjusted the parameter in this way, the test set is not independent anymore. This is identical to the corresponding problem in training on the *training* set: once we have chosen the function to minimize the training error, the latter no longer provides an unbiased estimate of the test error. Overfitting occurs much faster on the training set, however, than it does on the test set. This is usually due to the fact that the number of tuning parameters of a learning machine is much smaller than the total number of parameters, and thus the capacity tends to be smaller. For instance, an SVM for pattern recognition typically has two tuning parameters, and optimizes m weight parameters (for a training set size of m). See also Problem 5.3 and [461].

0 or 1 (provided we have a ± 1-valued function f), and all examples are drawn independently. In the language of probability theory, we are faced with *Bernoulli trials*. The $\xi_1, \ldots, \xi_m$ are independently sampled from a random variable

$$\xi := \frac{1}{2}|f(x) - y|. \tag{5.5}$$

Chernoff Bound

A famous inequality due to Chernoff [107] characterizes how the empirical mean $\frac{1}{m}\sum_{i=1}^m \xi_i$ converges to the expected value (or expectation) of ξ, denoted by $\mathbf{E}(\xi)$:

$$\mathrm{P}\left\{\left|\frac{1}{m}\sum_{i=1}^m \xi_i - \mathbf{E}(\xi)\right| \geq \epsilon\right\} \leq 2\exp(-2m\epsilon^2) \tag{5.6}$$

Note that the P refers to the probability of getting a sample $\xi_1, \ldots, \xi_m$ with the property $\left|\frac{1}{m}\sum_{i=1}^m \xi_i - \mathbf{E}(\xi)\right| \geq \epsilon$. Mathematically speaking, P strictly refers to a so-called *product* measure (cf. (B.11)). We will presently avoid further mathematical detail; more information can be found in Appendix B.

In some instances, we will use a more general bound, due to Hoeffding (Theorem 5.1). Presently, we formulate and prove a special case of the Hoeffding bound, which implies (5.6). Note that in the following statement, the ξ_i are no longer restricted to take values in $\{0, 1\}$.

Theorem 5.1 (Hoeffding [244]) *Let ξ_i, $i \in [m]$ be m independent instances of a bounded random variable ξ, with values in $[a, b]$. Denote their average by $Q_m = \frac{1}{m}\sum_i \xi_i$. Then for any $\epsilon > 0$,*

Hoeffding Bound

$$\left.\begin{array}{l} \mathrm{P}\{Q_m - \mathbf{E}(\xi) \geq \epsilon\} \\ \mathrm{P}\{\mathbf{E}(\xi) - Q_m \geq \epsilon\} \end{array}\right\} \leq \exp\left(-\frac{2m\epsilon^2}{(b-a)^2}\right). \tag{5.7}$$

The proof is carried out by using a technique commonly known as Chernoff's bounding method [107]. The proof technique is widely applicable, and generates bounds such as Bernstein's inequality [44] (exponential bounds based on the variance of random variables), as well as concentration-of-measure inequalities (see, e.g., [356, 66]). Readers not interested in the technical details underlying laws of large numbers may want to skip the following discussion.

We start with an auxiliary inequality.

Lemma 5.2 (Markov's Inequality (e.g., [136])) *Denote by ξ a nonnegative random variable with distribution* P. *Then for all $\lambda > 0$, the following inequality holds:*

$$\mathrm{P}\{\xi \geq \lambda \mathbf{E}(\xi)\} \leq \frac{1}{\lambda}. \tag{5.8}$$

Proof Using the definition of $\mathbf{E}(\xi)$, we have

$$\mathbf{E}(\xi) = \int_0^\infty \xi dP(\xi) \geq \int_{\lambda\mathbf{E}(\xi)}^\infty \xi dP(\xi) \geq \lambda\mathbf{E}(\xi)\int_{\lambda\mathbf{E}(\xi)}^\infty dP(\xi) = \lambda\mathbf{E}(\xi)\mathrm{P}\{\xi \geq \lambda\mathbf{E}(\xi)\}.$$

■

Proof of Theorem 5.1. Without loss of generality, we assume that $\mathbf{E}(\xi) = 0$ (otherwise simply define a random variable $\bar{\xi} := \xi - \mathbf{E}(\xi)$ and use the latter in the proof). Chernoff's bounding method consists in transforming a random variable ξ into $\exp(s\xi)$ ($s > 0$), and applying Markov's inequality to it. Depending on ξ, we can obtain different bounds. In our case, we use

$$\mathrm{P}\{\xi \geq \epsilon\} = \mathrm{P}\left\{\exp(s\xi) \geq \exp(s\epsilon)\right\} \leq e^{-s\epsilon}\mathbf{E}\left[\exp(s\xi)\right] \tag{5.9}$$

$$= e^{-s\epsilon}\mathbf{E}\left[\exp\left(\frac{s}{m}\sum_{i=1}^{m}\xi_i\right)\right] \leq e^{-s\epsilon}\prod_{i=1}^{m}\mathbf{E}\left[\exp\left(\frac{s}{m}\xi_i\right)\right]. \tag{5.10}$$

In (5.10), we exploited the fact that for positive random variables $\mathbf{E}\left[\prod_i \xi_i\right] \leq \prod_i \mathbf{E}\left[\xi_i\right]$. Since the inequality holds independent of the choice of s, we may minimize over s to obtain a bound that is as tight as possible. To this end, we transform the expectation over $\exp\left(\frac{s}{m}\xi_i\right)$ into something more amenable. The derivation is rather technical; thus we state without proof [244]: $\mathbf{E}\left[\exp(\frac{s}{m}\xi_i)\right] \leq \exp\left(\frac{s^2(b-a)^2}{8m^2}\right)$. From this, we conclude that the optimal value of s is given by $s = \frac{4m\epsilon}{(b-a)^2}$. Substituting this value into the right hand side of (5.10) proves the bound. ■

Let us now return to (5.6). Substituting (5.5) into (5.6), we have a bound which states how likely it is that for a given function f, the empirical risk is close to the actual risk,

$$\mathrm{P}\{|R_{\mathrm{emp}}[f] - R[f]| \geq \epsilon\} \leq 2\exp(-2m\epsilon^2). \tag{5.11}$$

Using Hoeffding's inequality, a similar bound can be given for the case of regression estimation, provided the loss $c(x, y, f(x))$ is bounded.

For any fixed function, the training error thus provides an unbiased estimate of the test error. Moreover, the convergence (in probability) $R_{\mathrm{emp}}[f] \to R[f]$ as $m \to \infty$ is exponentially fast in the number of training examples.[2] Although this sounds just about as good as we could possibly have hoped, there is one caveat: a crucial property of both the Chernoff and the Hoeffding bound is that they are probabilistic in nature. They state that the probability of a large deviation between test error and training error of f is small; the larger the sample size m, the smaller the probability. Granted, they do not rule out the presence of cases where the deviation is large, and our learning machine will have many functions that it can implement. Could there be a function for which things go wrong? It appears that

2. *Convergence in probability*, denoted as

$$|R_{\mathrm{emp}}[f] - R[f]| \xrightarrow{\mathrm{P}} 0 \text{ as } m \to \infty,$$

means that for all $\epsilon > 0$, we have

$$\lim_{m\to\infty} \mathrm{P}\{|R_{\mathrm{emp}}[f] - R[f]| > \epsilon\} = 0.$$

we would be very unlucky for this to occur *precisely* for the function f chosen by empirical risk minimization.

At first sight, it seems that empirical risk minimization should work — in contradiction to our lengthy explanation in the last section, arguing that we have to do more than that. What is the catch?

5.3 When Does Learning Work: the Question of Consistency

It turns out that in the last section, we were too sloppy. When we find a function f by choosing it to minimize the training error, we are no longer looking at independent Bernoulli trials. We are actually choosing f such that the mean of the ξ_i is as small as possible. In this sense, we are actively looking for the worst case, for a function which is very atypical, with respect to the average loss (i.e., the empirical risk) that it will produce.

We should thus state more clearly what it is that we actually need for empirical risk minimization to work. This is best expressed in terms of a notion that statisticians call *consistency*. It amounts to saying that as the number of examples m tends to infinity, we want the function f^m that minimizes $R_{\text{emp}}[f]$ (note that f^m need not be unique), to lead to a test error which converges to the lowest achievable value. In other words, f^m is asymptotically as good as whatever we could have done if we were able to directly minimize $R[f]$ (which we cannot, as we do not even know it). In addition, consistency requires that asymptotically, the training and the test error of f^m be identical.[3]

Consistency

It turns out that *without restricting the set of admissible functions*, empirical risk minimization is not consistent. The main insight of VC (Vapnik-Chervonenkis) theory is that actually, the *worst case* over all functions that the learning machine can implement determines the consistency of empirical risk minimization. In other words, we need a version of the law of large numbers which is *uniform* over all functions that the learning machine can implement.

5.4 Uniform Convergence and Consistency

The present section will explain how consistency can be characterized by a uniform convergence condition on the set of functions $\mathcal{F}$ that the learning machine can implement. Figure 5.2 gives a simplified depiction of the question of consistency. Both the empirical risk and the actual risk are drawn as functions of f. For

3. We refrain from giving a more formal definition of consistency, the reason being that there are some caveats to this classical definition of consistency; these would necessitate a discussion leading us away from the main thread of the argument. For the precise definition of the required notion of "nontrivial consistency," see [561].

we would be very unlucky for this to occur *precisely* for the function f chosen by empirical risk minimization.

At first sight, it seems that empirical risk minimization should work — in contradiction to our lengthy explanation in the last section, arguing that we have to do more than that. What is the catch?

5.3 When Does Learning Work: the Question of Consistency

It turns out that in the last section, we were too sloppy. When we find a function f by choosing it to minimize the training error, we are no longer looking at independent Bernoulli trials. We are actually choosing f such that the mean of the ξ_i is as small as possible. In this sense, we are actively looking for the worst case, for a function which is very atypical, with respect to the average loss (i.e., the empirical risk) that it will produce.

We should thus state more clearly what it is that we actually need for empirical risk minimization to work. This is best expressed in terms of a notion that statisticians call *consistency*. It amounts to saying that as the number of examples m tends to infinity, we want the function f^m that minimizes $R_{\text{emp}}[f]$ (note that f^m need not be unique), to lead to a test error which converges to the lowest achievable value. In other words, f^m is asymptotically as good as whatever we could have done if we were able to directly minimize $R[f]$ (which we cannot, as we do not even know it). In addition, consistency requires that asymptotically, the training and the test error of f^m be identical.[3]

Consistency

It turns out that *without restricting the set of admissible functions*, empirical risk minimization is not consistent. The main insight of VC (Vapnik-Chervonenkis) theory is that actually, the *worst case* over all functions that the learning machine can implement determines the consistency of empirical risk minimization. In other words, we need a version of the law of large numbers which is *uniform* over all functions that the learning machine can implement.

5.4 Uniform Convergence and Consistency

The present section will explain how consistency can be characterized by a uniform convergence condition on the set of functions $\mathcal{F}$ that the learning machine can implement. Figure 5.2 gives a simplified depiction of the question of consistency. Both the empirical risk and the actual risk are drawn as functions of f. For

3. We refrain from giving a more formal definition of consistency, the reason being that there are some caveats to this classical definition of consistency; these would necessitate a discussion leading us away from the main thread of the argument. For the precise definition of the required notion of "nontrivial consistency," see [561].

we would be very unlucky for this to occur *precisely* for the function f chosen by empirical risk minimization.

At first sight, it seems that empirical risk minimization should work — in contradiction to our lengthy explanation in the last section, arguing that we have to do more than that. What is the catch?

5.3 When Does Learning Work: the Question of Consistency

It turns out that in the last section, we were too sloppy. When we find a function f by choosing it to minimize the training error, we are no longer looking at independent Bernoulli trials. We are actually choosing f such that the mean of the ξ_i is as small as possible. In this sense, we are actively looking for the worst case, for a function which is very atypical, with respect to the average loss (i.e., the empirical risk) that it will produce.

We should thus state more clearly what it is that we actually need for empirical risk minimization to work. This is best expressed in terms of a notion that statisticians call *consistency*. It amounts to saying that as the number of examples m tends to infinity, we want the function f^m that minimizes $R_{\text{emp}}[f]$ (note that f^m need not be unique), to lead to a test error which converges to the lowest achievable value. In other words, f^m is asymptotically as good as whatever we could have done if we were able to directly minimize $R[f]$ (which we cannot, as we do not even know it). In addition, consistency requires that asymptotically, the training and the test error of f^m be identical.[3]

Consistency

It turns out that *without restricting the set of admissible functions*, empirical risk minimization is not consistent. The main insight of VC (Vapnik-Chervonenkis) theory is that actually, the *worst case* over all functions that the learning machine can implement determines the consistency of empirical risk minimization. In other words, we need a version of the law of large numbers which is *uniform* over all functions that the learning machine can implement.

5.4 Uniform Convergence and Consistency

The present section will explain how consistency can be characterized by a uniform convergence condition on the set of functions $\mathcal{F}$ that the learning machine can implement. Figure 5.2 gives a simplified depiction of the question of consistency. Both the empirical risk and the actual risk are drawn as functions of f. For

3. We refrain from giving a more formal definition of consistency, the reason being that there are some caveats to this classical definition of consistency; these would necessitate a discussion leading us away from the main thread of the argument. For the precise definition of the required notion of "nontrivial consistency," see [561].

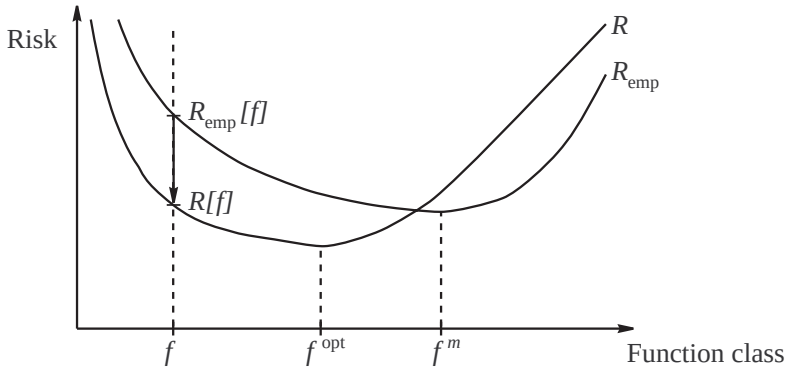


Figure 5.2 Simplified depiction of the convergence of empirical risk to actual risk. The x-axis gives a one-dimensional representation of the function class; the y axis denotes the risk (error). For each *fixed* function f, the law of large numbers tells us that as the sample size goes to infinity, the empirical risk $R_{\text{emp}}[f]$ converges towards the true risk $R[f]$ (indicated by the downward arrow). This does not imply, however, that in the limit of infinite sample sizes, the minimizer of the empirical risk, f^m, will lead to a value of the risk that is as good as the best attainable risk, $R[f^{\text{opt}}]$ *(consistency)*. For the latter to be true, we require convergence of $R_{\text{emp}}[f]$ towards $R[f]$ to be uniform over all functions that the learning machines can implement (see text).

simplicity, we have summarized all possible functions f by a single axis of the plot. Empirical risk minimization consists in picking the f that yields the minimal value of R_{emp}. If it is consistent, then the minimum of R_{emp} converges to that of R in probability. Let us denote the minimizer of R by f^{opt}, satisfying

$$R[f] - R[f^{\text{opt}}] \geq 0 \tag{5.12}$$

for all $f \in \mathcal{F}$. This is the optimal choice that we could make, given complete knowledge of the distribution P.[4] Similarly, since f^m minimizes the empirical risk, we have

$$R_{\text{emp}}[f] - R_{\text{emp}}[f^m] \geq 0, \tag{5.13}$$

for all $f \in \mathcal{F}$. Being true for all $f \in \mathcal{F}$, (5.12) and (5.13) hold in particular for f^m and f^{opt}. If we substitute the former into (5.12) and the latter into (5.13), we obtain

$$R[f^m] - R[f^{\text{opt}}] \geq 0, \tag{5.14}$$

and

$$R_{\text{emp}}[f^{\text{opt}}] - R_{\text{emp}}[f^m] \geq 0. \tag{5.15}$$

4. As with f^m, f^{opt} need not be unique.

The sum of these two inequalities satisfies

$$\begin{aligned} 0 &\leq R[f^m] - R[f^{\text{opt}}] + R_{\text{emp}}[f^{\text{opt}}] - R_{\text{emp}}[f^m] \\ &= R[f^m] - R_{\text{emp}}[f^m] + R_{\text{emp}}[f^{\text{opt}}] - R[f^{\text{opt}}] \\ &\leq \sup_{f \in \mathcal{F}} \big(R[f] - R_{\text{emp}}[f]\big) + R_{\text{emp}}[f^{\text{opt}}] - R[f^{\text{opt}}]. \end{aligned} \tag{5.16}$$

Let us first consider the second half of the right hand side. Due to the law of large numbers, we have convergence in probability, i.e., for all $\epsilon > 0$,

$$|R_{\text{emp}}[f^{\text{opt}}] - R[f^{\text{opt}}]| \xrightarrow{\text{P}} 0 \text{ as } m \to \infty. \tag{5.17}$$

This holds true since f^{opt} is a fixed function, which is independent of the training sample (see (5.11)).

The important conclusion is that if the empirical risk converges to the actual risk one-sided *uniformly*, over all functions that the learning machine can implement,

Uniform Convergence of Risk

$$\sup_{f \in \mathcal{F}} (R[f] - R_{\text{emp}}[f]) \xrightarrow{\text{P}} 0 \;\text{ as }\; m \to \infty, \tag{5.18}$$

then the left hand sides of (5.14) and (5.15) will likewise converge to 0;

$$R[f^m] - R[f^{\text{opt}}] \xrightarrow{\text{P}} 0, \tag{5.19}$$

$$R_{\text{emp}}[f^{\text{opt}}] - R_{\text{emp}}[f^m] \xrightarrow{\text{P}} 0. \tag{5.20}$$

As argued above, (5.17) is not always true for f^m, since f^m is chosen to minimize R_{emp}, and thus depends on the sample. Assuming that (5.18) holds true, however, then (5.19) and (5.20) imply that in the limit, $R[f^m]$ cannot be larger than $R_{\text{emp}}[f^m]$. One-sided uniform convergence on $\mathcal{F}$ is thus a sufficient condition for consistency of the empirical risk minimization over $\mathcal{F}$.[5]

What about the other way round? Is one-sided uniform convergence also a *necessary* condition? Part of the mathematical beauty of VC theory lies in the fact that this is the case. We cannot go into the necessary details to prove this [571, 561, 562], and only state the main result. Note that this theorem uses the notion of nontrivial consistency that we already mentioned briefly in footnote 3. In a nutshell, this concept requires that the induction principle be consistent even after the "best" functions have been removed. Nontrivial consistency thus rules out, for instance, the case in which the problem is trivial, due to the existence of a function which uniformly does better than all other functions. To understand this, assume that there exists such a function. Since this function is uniformly better than all others, we can already select this function (using ERM) from *one* (arbitrary) data point. Hence the method would be trivially consistent, no matter what the

5. Note that the onesidedness of the convergence comes from the fact that we only require consistency of empirical risk *minimization*. If we required the same for empirical risk *maximization*, then we would end up with standard uniform convergence, and the parentheses in (5.18) would be replaced with modulus signs.

rest of the function class looks like. Having one function which gets picked as soon as we have seen one data point would essentially void the inherently *asymptotic* notion of consistency.

Theorem 5.3 (Vapnik & Chervonenkis (e.g., [562])) *One-sided uniform convergence in probability,*

$$\lim_{m\to\infty} \mathrm{P}\{\sup_{f\in\mathcal{F}}(R[f] - R_{\mathrm{emp}}[f]) > \epsilon\} = 0, \tag{5.21}$$

for all $\epsilon > 0$*, is a* necessary and sufficient *condition for nontrivial consistency of empirical risk minimization.*

As explained above, consistency, and thus learning, crucially depends on the set of functions. In Section 5.1, we gave an example where we considered the set of all possible functions, and showed that learning was impossible. The dependence of learning on the set of functions has now returned in a different guise: the condition of uniform convergence will crucially depend on the set of functions for which it must hold.

The abstract characterization in Theorem 5.3 of consistency as a uniform convergence property, whilst theoretically intriguing, is not all that useful in practice. We do not want to check some fairly abstract convergence property every time we want to use a learning machine. Therefore, we next address whether there are properties of learning machines, i.e., of sets of functions, which *ensure* uniform convergence of risks.

5.5 How to Derive a VC Bound

We now take a closer look at the subject of Theorem 5.3; the probability

$$\mathrm{P}\{\sup_{f\in\mathcal{F}}(R[f] - R_{\mathrm{emp}}[f]) > \epsilon\}. \tag{5.22}$$

We give a simplified account, drawing from the expositions of [561, 562, 415, 238]. We do not aim to describe or even develop the theory to the extent that would be necessary to give precise bounds for SVMs, say. Instead, our goal will be to convey central insights rather than technical details. For more complete treatments geared specifically towards SVMs, cf. [562, 491, 24]. We focus on the case of pattern recognition; that is, on functions taking values in $\{\pm 1\}$.

Two tricks are needed along the way: the *union bound* and the method of *symmetrization by a ghost sample*.

5.5.1 The Union Bound

Suppose the set $\mathcal{F}$ consists of two functions, f_1 and f_2. In this case, uniform convergence of risk trivially follows from the law of large numbers, which holds

rest of the function class looks like. Having one function which gets picked as soon as we have seen one data point would essentially void the inherently *asymptotic* notion of consistency.

Theorem 5.3 (Vapnik & Chervonenkis (e.g., [562])) *One-sided uniform convergence in probability,*

$$\lim_{m\to\infty} \mathrm{P}\{\sup_{f\in\mathcal{F}}(R[f] - R_{\mathrm{emp}}[f]) > \epsilon\} = 0, \tag{5.21}$$

for all $\epsilon > 0$*, is a* necessary and sufficient *condition for nontrivial consistency of empirical risk minimization.*

As explained above, consistency, and thus learning, crucially depends on the set of functions. In Section 5.1, we gave an example where we considered the set of all possible functions, and showed that learning was impossible. The dependence of learning on the set of functions has now returned in a different guise: the condition of uniform convergence will crucially depend on the set of functions for which it must hold.

The abstract characterization in Theorem 5.3 of consistency as a uniform convergence property, whilst theoretically intriguing, is not all that useful in practice. We do not want to check some fairly abstract convergence property every time we want to use a learning machine. Therefore, we next address whether there are properties of learning machines, i.e., of sets of functions, which *ensure* uniform convergence of risks.

5.5 How to Derive a VC Bound

We now take a closer look at the subject of Theorem 5.3; the probability

$$\mathrm{P}\{\sup_{f\in\mathcal{F}}(R[f] - R_{\mathrm{emp}}[f]) > \epsilon\}. \tag{5.22}$$

We give a simplified account, drawing from the expositions of [561, 562, 415, 238]. We do not aim to describe or even develop the theory to the extent that would be necessary to give precise bounds for SVMs, say. Instead, our goal will be to convey central insights rather than technical details. For more complete treatments geared specifically towards SVMs, cf. [562, 491, 24]. We focus on the case of pattern recognition; that is, on functions taking values in $\{\pm 1\}$.

Two tricks are needed along the way: the *union bound* and the method of *symmetrization by a ghost sample*.

5.5.1 The Union Bound

Suppose the set $\mathcal{F}$ consists of two functions, f_1 and f_2. In this case, uniform convergence of risk trivially follows from the law of large numbers, which holds

for each of the two. To see this, let

$$C_\epsilon^i := \{(x_1, y_1), \ldots, (x_m, y_m) | (R[f_i] - R_{\text{emp}}[f_i]) > \epsilon\} \tag{5.23}$$

denote the set of samples for which the risks of f_i differ by more than ϵ. Then, by definition, we have

$$\mathrm{P}\{\sup_{f \in \mathcal{F}}(R[f] - R_{\text{emp}}[f]) > \epsilon\} = \mathrm{P}(C_\epsilon^1 \cup C_\epsilon^2). \tag{5.24}$$

The latter, however, can be rewritten as

$$\mathrm{P}(C_\epsilon^1 \cup C_\epsilon^2) = \mathrm{P}(C_\epsilon^1) + \mathrm{P}(C_\epsilon^2) - \mathrm{P}(C_\epsilon^1 \cap C_\epsilon^2) \leq \mathrm{P}(C_\epsilon^1) + \mathrm{P}(C_\epsilon^2), \tag{5.25}$$

where the last inequality follows from the fact that P is nonnegative. Similarly, if $\mathcal{F} = \{f_1, \ldots, f_n\}$, we have

$$\mathrm{P}\{\sup_{f \in \mathcal{F}}(R[f] - R_{\text{emp}}[f]) > \epsilon\} = \mathrm{P}(C_\epsilon^1 \cup \ldots \cup C_\epsilon^n) \leq \sum_{i=1}^{n} \mathrm{P}(C_\epsilon^i). \tag{5.26}$$

Union Bound

This inequality is called the *union bound*. As it is a crucial step in the derivation of risk bounds, it is worthwhile to emphasize that it becomes an equality if and only if all the events involved are *disjoint*. In practice, this is rarely the case, and we therefore lose a lot when applying (5.26). It is a step with a large "slack."

Nevertheless, when $\mathcal{F}$ is finite, we may simply apply the law of large numbers (5.11) for each individual $\mathrm{P}(C_\epsilon^i)$, and the sum in (5.26) then leads to a constant factor n on the right hand side of the bound — it does not change the exponentially fast convergence of the empirical risk towards the actual risk. In the next section, we describe an ingenious trick used by Vapnik and Chervonenkis, to reduce the infinite case to the finite one. It consists of introducing what is sometimes called a *ghost sample*.

5.5.2 Symmetrization

The central observation in this section is that we can bound (5.22) in terms of a probability of an event referring to a *finite* function class. Note first that the empirical risk term in (5.22) effectively refers only to a finite function class: for any given training sample of m points $x_1, \ldots, x_m$, the functions of $\mathcal{F}$ can take at most 2^m different values $y_1, \ldots, y_m$ (recall that the y_i take values only in $\{\pm 1\}$). In addition, the probability that the empirical risk differs from the actual risk by more than ϵ, can be bounded by the twice the probability that it differs from the empirical risk on a *second* sample of size m by more than $\epsilon/2$.

Lemma 5.4 (Symmetrization (Vapnik & Chervonenkis) (e.g. [559])) *For $m\epsilon^2 \geq 2$, we have*

Symmetrization

$$\mathrm{P}\{\sup_{f \in \mathcal{F}}(R[f] - R_{\text{emp}}[f]) > \epsilon\} \leq 2\mathrm{P}\{\sup_{f \in \mathcal{F}}(R_{\text{emp}}[f] - R'_{\text{emp}}[f]) > \epsilon/2\}. \tag{5.27}$$

Here, the first P *refers to the distribution of iid samples of size m, while the second one*

refers to iid samples of size $2m$*. In the latter case,* R_{emp} *measures the loss on the first half of the sample, and* R'_{emp} *on the second half.*

Although we do not prove this result, it should be fairly plausible: if the empirical error rates on two independent m-samples are close to each other, then they should also be close to the true error rate.

5.5.3 The Shattering Coefficient

The main result of Lemma 5.4 is that it implies, for the purpose of bounding (5.22), that the function class $\mathcal{F}$ is effectively finite: restricted to the $2m$ points appearing on the right hand side of (5.27), it has *at most* 2^{2m} elements. This is because only the outputs of the functions on the patterns of the sample count, and there are $2m$ patterns with two possible outputs, ± 1. The number of effectively different functions can be smaller than 2^{2m}, however; and for our purposes, this is the case that will turn out to be interesting.

Let $Z_{2m} := \big((x_1, y_1), \ldots, (x_{2m}, y_{2m})\big)$ be the given $2m$-sample. Denote by $\mathcal{N}(\mathcal{F}, Z_{2m})$ the cardinality of $\mathcal{F}$ when restricted to $\{x_1, \ldots, x_{2m}\}$, that is, the number of functions from $\mathcal{F}$ that can be distinguished from their values on $\{x_1, \ldots, x_{2m}\}$. Let us, moreover, denote the maximum (over all possible choices of a $2m$-sample) number of functions that can be distinguished in this way as $\mathcal{N}(\mathcal{F}, 2m)$.

Shattering Coefficient

The function $\mathcal{N}(\mathcal{F}, m)$ is referred to as the *shattering coefficient*, or in the more general case of regression estimation, the *covering number* of $\mathcal{F}$.[6] In the case of pattern recognition, which is what we are currently looking at, $\mathcal{N}(\mathcal{F}, m)$ has a particularly simple interpretation: it is the number of different outputs $(y_1, \ldots, y_m)$ that the functions in $\mathcal{F}$ can achieve on samples of a given size.[7] In other words, it simply measures the *number of ways that the function class can separate the patterns into two classes*. Whenever $\mathcal{N}(\mathcal{F}, m) = 2^m$, all possible separations can be implemented by functions of the class. In this case, the function class is said to *shatter* m points. Note that this means that there *exists* a set of m patterns which can be separated in all possible ways — it does not mean that this applies to *all* sets of m patterns.

Shattering

5.5.4 Uniform Convergence Bounds

Let us now take a closer look at the probability that for a $2m$-sample Z_{2m} drawn iid from P, we get a one-sided uniform deviation larger than $\epsilon/2$ (cf. (5.27)),

$$\mathrm{P}\{\sup_{f \in \mathcal{F}} (R_{\text{emp}}[f] - R'_{\text{emp}}[f]) > \epsilon/2\}. \tag{5.28}$$

6. In regression estimation, the covering number also depends on the accuracy within which we are approximating the function class, and on the loss function used; see Section 12.4 for more details.

7. Using the zero-one loss $c(x, y, f(x)) = 1/2|f(x) - y| \in \{0, 1\}$, it also equals the number of different loss vectors $(c(x_1, y_1, f(x_1)), \ldots, c(x_m, y_m, f(x_m)))$.

The basic idea now is to pick a maximal set of functions $\{f_1, \ldots, f_{\mathcal{N}(\mathcal{F}, Z_{2m})}\}$ that can be distinguished based on their values on Z_{2m}, then use the union bound, and finally bound each term using the Chernoff inequality. However, the fact that the f_i depend on the sample Z_{2m} will make things somewhat more complicated. To deal with this, we have to introduce an auxiliary step of randomization, using a uniform distibution over permutations σ of the $2m$-sample Z_{2m}.

Let us denote the empirical risks on the two halves of the sample after the permutation σ by $R^{\sigma}_{\mathrm{emp}}[f]$ and $R'^{\sigma}_{\mathrm{emp}}[f]$, respectively. Since the $2m$-sample is iid, the permutation does not affect (5.28). We may thus instead consider

$$\mathrm{P}_{Z_{2m},\sigma}\{\sup_{f\in\mathcal{F}}(R^{\sigma}_{\mathrm{emp}}[f] - R'^{\sigma}_{\mathrm{emp}}[f]) > \epsilon/2\}, \tag{5.29}$$

where the subscripts of P were added to clarify what the distribution refers to. We next rewrite this as

$$\int_{(\mathcal{X}\times\{\pm1\})^{2m}} \mathrm{P}_{\sigma|Z_{2m}}\{\sup_{f\in\mathcal{F}|_{Z_{2m}}} (R^{\sigma}_{\mathrm{emp}}[f] - R'^{\sigma}_{\mathrm{emp}}[f]) > \epsilon/2 \mid Z_{2m}\}\, d\mathrm{P}(Z_{2m}). \tag{5.30}$$

We can now express the event $C_\epsilon := \{\sigma | \sup_{f\in\mathcal{F}|_{Z_{2m}}} (R^{\sigma}_{\mathrm{emp}}[f] - R'^{\sigma}_{\mathrm{emp}}[f]) > \epsilon/2\}$ as

$$C_\epsilon = \bigcup_{n=1}^{\mathcal{N}(\mathcal{F},Z_{2m})} C_\epsilon(f_n), \tag{5.31}$$

where the events $C_\epsilon(f_n) := \{\sigma | (R^{\sigma}_{\mathrm{emp}}[f_n] - R'^{\sigma}_{\mathrm{emp}}[f_n]) > \epsilon/2\}$ refer to individual functions f_n chosen such that $\left(\bigcup_n \{f_n\}\right)|_{Z_{2m}} = \mathcal{F}|_{Z_{2m}}$. Note that the functions f_n may be considered as fixed, since we have conditioned on Z_{2m}.

We are now in a position to appeal to the classical law of large numbers. Our random experiment consists of drawing σ from the uniform distribution over all permutations of $2m$-samples. This turns our sequence of losses $\xi^{\sigma}_i = \frac{1}{2}|f(x^{\sigma}_i) - y^{\sigma}_i|$ $(i = 1, \ldots, 2m)$ into an iid sequence of independent Bernoulli trials. We then apply a modified Chernoff inequality to bound the probability of each event $C_\epsilon(f_n)$. It states that given a $2m$-sample of Bernoulli trials, we have (see Problem 5.4)

$$\mathrm{P}\left\{\frac{1}{m}\sum_{i=1}^{m}\xi_i - \frac{1}{m}\sum_{i=m+1}^{2m}\xi_i \geq \epsilon\right\} \leq 2\,\exp\left(-\frac{m\epsilon^2}{2}\right). \tag{5.32}$$

For our present problem, we thus obtain

$$\mathrm{P}_{\sigma|Z_{2m}}(C_\epsilon(f_n)) \leq 2\,\exp\left(-\frac{m\epsilon^2}{8}\right), \tag{5.33}$$

independent of f_n. We next use the union bound to get a bound on the probability of the event C_ϵ defined in (5.31). We obtain a sum over $\mathcal{N}(\mathcal{F}, Z_{2m})$ identical terms of the form (5.33). Hence (5.30) (and (5.29)) can be bounded from above by

$$\begin{aligned}&\int_{(\mathcal{X}\times\{\pm1\})^{2m}} \mathcal{N}(\mathcal{F}, Z_{2m})\, 2\,\exp\left(-\frac{m\epsilon^2}{8}\right) d\mathrm{P}(Z_{2m})\\ &= 2\,\mathbf{E}\left[\mathcal{N}(\mathcal{F}, Z_{2m})\right]\exp\left(-\frac{m\epsilon^2}{8}\right),\end{aligned} \tag{5.34}$$

where the expectation is taken over the random drawing of Z_{2m}. The last step is to combine this with Lemma 5.4, to obtain

$$\begin{aligned} \mathrm{P}\{\sup_{f\in\mathcal{F}}(R[f]-R_{\mathrm{emp}}[f])>\epsilon\} &\leq 4\,\mathbf{E}\left[\mathcal{N}(\mathcal{F},Z_{2m})\right]\,\exp\left(-\frac{m\epsilon^2}{8}\right) \\ &= 4\,\exp\left(\ln\mathbf{E}\left[\mathcal{N}(\mathcal{F},Z_{2m})\right]-\frac{m\epsilon^2}{8}\right). \end{aligned} \tag{5.35}$$

Inequality of Vapnik-Chervonenkis Type

We conclude that provided $\mathbf{E}\left[\mathcal{N}(\mathcal{F},Z_{2m})\right]$ does not grow exponentially in m (i.e., $\ln\mathbf{E}\left[\mathcal{N}(\mathcal{F},Z_{2m})\right]$ grows sublinearly), it is actually possible to make nontrivial statements about the *test* error of learning machines.

The above reasoning is essentially the VC style analysis. Similar bounds can be obtained using a strategy which is more common in the field of empirical processes, first proving that $\sup_f(R[f]-R_{\mathrm{emp}}[f])$ is concentrated around its mean [554, 14].

5.5.5 Confidence Intervals

It is sometimes useful to rewrite (5.35) such that we specify the probability with which we want the bound to hold, and then get the confidence interval, which tells us how close the risk should be to the empirical risk. This can be achieved by setting the right hand side of (5.35) equal to some $\delta>0$, and then solving for ϵ. As a result, we get the statement that with a probability at least $1-\delta$,

Risk Bound

$$R[f]\leq R_{\mathrm{emp}}[f]+\sqrt{\frac{8}{m}\left(\ln\mathbf{E}\left[\mathcal{N}(\mathcal{F},Z_{2m})\right]+\ln\frac{4}{\delta}\right)}. \tag{5.36}$$

Note that this bound holds independent of f; in particular, it holds for the function f^m minimizing the empirical risk. This is not only a strength, but also a weakness in the bound. It is a strength since many learning machines do not truly minimize the empirical risk, and the bound thus holds for them, too. It is a weakness since by taking into account more information on which function we are interested in, one could hope to get more accurate bounds. We will return to this issue in Section 12.1.

Bounds like (5.36) can be used to justify induction principles different from the empirical risk minimization principle. Vapnik and Chervonenkis [569, 559] proposed minimizing the right hand side of these bounds, rather than just the empirical risk. The confidence term, in the present case, $\sqrt{\frac{8}{m}\left(\ln\mathbf{E}\left[\mathcal{N}(\mathcal{F},Z_{2m})\right]+\ln\frac{4}{\delta}\right)}$, then ensures that the chosen function, denoted f_*, not only leads to a small risk, but also comes from a function class with small capacity.

The capacity term is a property of the function class $\mathcal{F}$, and not of any individual function f. Thus, the bound cannot simply be minimized over choices of f. Instead, we introduce a so-called *structure* on $\mathcal{F}$, and minimize over the choice of the structure. This leads to an induction principle called *structural risk minimization*. We leave out the technicalities involved [559, 136, 562]. The main idea is depicted in Figure 5.3.

Structural Risk Minimization

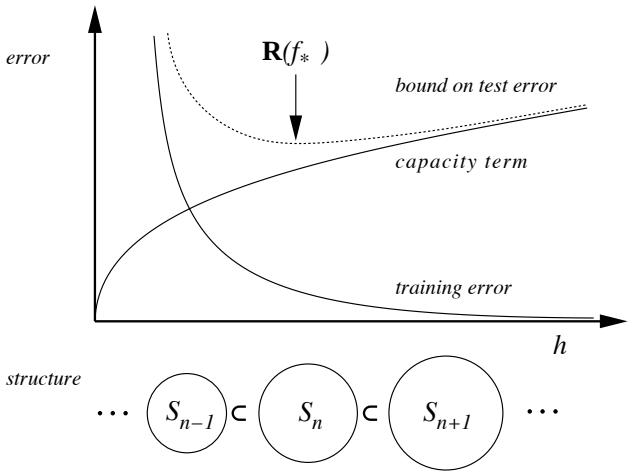


Figure 5.3 Graphical depiction of the structural risk minimization (SRM) induction principle. The function class is decomposed into a nested sequence of subsets of increasing size (and thus, of increasing capacity). The SRM principle picks a function f_* which has small training error, and comes from an element of the structure that has low capacity h, thus minimizing a risk bound of type (5.36).

For practical purposes, we usually employ bounds of the type (5.36) as a guideline for coming up with risk functionals (see Section 4.1). Often, the risk functionals form a compromise between quantities that *should* be minimized from a statistical point of view, and quantities that *can* be minimized efficiently (cf. Problem 5.7).

There exists a large number of bounds similar to (5.35) and its alternative form (5.36). Differences occur in the constants, both in front of the exponential and in its exponent. The bounds also differ in the exponent of ϵ — in some cases, by a factor greater than 2. For instance, if a training error of zero is achievable, we can use Bernstein's inequality instead of Chernoff's result, which leads to ϵ rather than ϵ^2. For further details, cf. [136, 562, 492, 238]. Finally, the bounds differ in the way they measure capacity. So far, we have used covering numbers, but this is not the only method.

5.5.6 The VC Dimension and Other Capacity Concepts

So far, we have formulated the bounds in terms of the so-called *annealed entropy* $\ln \mathbf{E}[\mathcal{N}(\mathcal{F}, Z_{2m})]$. This led to statements that depend on the distribution and thus can take into account characteristics of the problem at hand. The downside is that they are usually difficult to evaluate; moreover, in most problems, we do not have knowledge of the underlying distribution. However, a number of different capacity concepts, with different properties, can take the role of the term $\ln(\mathbf{E}[\mathcal{N}(\mathcal{F}, Z_{2m})])$ in (5.36).

- Given an example (x, y), $f \in \mathcal{F}$ causes a loss that we denote by $c(x, y, f(x)) := \frac{1}{2}|f(x) - y| \in \{0, 1\}$. For a larger sample $(x_1, y_1) \ldots, (x_m, y_m)$, the different functions

VC Entropy

$f \in \mathcal{F}$ lead to a *set* of loss vectors $\boldsymbol{\xi}_f = (c(x_1, y_1, f(x_1)), \ldots, c(x_m, y_m, f(x_m)))$, whose cardinality we denote by $\mathcal{N}\left(\mathcal{F}, (x_1, y_1) \ldots, (x_m, y_m)\right)$. The *VC entropy* is defined as

$$H_{\mathcal{F}}(m) = \mathbf{E}\left[\ln \mathcal{N}\left(\mathcal{F}, (x_1, y_1) \ldots, (x_m, y_m)\right)\right], \tag{5.37}$$

where the expectation is taken over the random generation of the m-sample $(x_1, y_1) \ldots, (x_m, y_m)$ from P.
One can show [562] that the convergence

$$\lim_{m\to\infty} H_{\mathcal{F}}(m)/m = 0, \tag{5.38}$$

is equivalent to uniform (two-sided) convergence of risk,

$$\lim_{m\to\infty} \mathrm{P}\{\sup_{f\in\mathcal{F}} |R[f] - R_{\mathrm{emp}}[f]| > \epsilon\} = 0, \tag{5.39}$$

for all $\epsilon > 0$. By Theorem 5.3, (5.39) thus implies consistency of empirical risk minimization.

Annealed Entropy

- If we exchange the expectation **E** and the logarithm in (5.37), we obtain the annealed entropy used above,

$$H^{\mathrm{ann}}_{\mathcal{F}}(m) = \ln \mathbf{E}\left[\mathcal{N}\left(\mathcal{F}, (x_1, y_1) \ldots, (x_m, y_m)\right)\right]. \tag{5.40}$$

Since the logarithm is a concave function, the annealed entropy is an upper bound on the VC entropy. Therefore, whenever the annealed entropy satisfies a condition of the form (5.38), the same automatically holds for the VC entropy.
One can show that the convergence

$$\lim_{m\to\infty} H^{\mathrm{ann}}_{\mathcal{F}}(m)/m = 0, \tag{5.41}$$

implies exponentially fast convergence [561],

$$P\{\sup_{f\in\mathcal{F}} |R[f] - R_{\mathrm{emp}}[f]| > \epsilon\} \leq 4\, \exp(((H^{\mathrm{ann}}_{\mathcal{F}}(2m)/m) - \epsilon^2) \cdot m). \tag{5.42}$$

It has recently been proven that in fact (5.41) is not only sufficient, but also necessary for this [66].

Growth Function

- We can obtain an upper bound on both entropies introduced so far, by taking a supremum over all possible samples, instead of the expectation. This leads to the *growth function*,

$$G_{\mathcal{F}}(m) = \max_{(x_1,y_1),\ldots,(x_m,y_m)\in\mathcal{X}\times\{\pm 1\}} \ln \mathcal{N}\left(\mathcal{F}, (x_1, y_1) \ldots, (x_m, y_m)\right). \tag{5.43}$$

Note that by definition, the growth function is the logarithm of the shattering coefficient, $G_{\mathcal{F}}(m) = \ln \mathcal{N}(\mathcal{F}, m)$.
The convergence

$$\lim_{m\to\infty} G_{\mathcal{F}}(m)/m = 0, \tag{5.44}$$

is necessary and sufficient for exponentially fast convergence of risk *for all underlying distributions* P.

■ The next step will be to summarize the main behavior of the growth function with a single number. If $\mathcal{F}$ is as rich as possible, so that for any sample of size m, the points can be chosen such that by using functions of the learning machine, they can be separated in all 2^m possible ways (i.e., they can be shattered), then

$$G_{\mathcal{F}}(m) = m \cdot \ln(2). \tag{5.45}$$

In this case, the convergence (5.44) does not take place, and learning will not generally be successful. What about the other case? Vapnik and Chervonenkis [567, 568] showed that either (5.45) holds true for all m, or there exists some *maximal* m for which (5.45) is satisfied. This number is called the *VC dimension* and is denoted by h. If the maximum does not exist, the VC dimension is said to be infinite.

VC Dimension

By construction, the VC dimension is thus the maximal number of points which can be shattered by functions in $\mathcal{F}$. It is possible to prove that for $m > h$ [568],

$$G_{\mathcal{F}}(m) \leq h \left(\ln \frac{m}{h} + 1 \right). \tag{5.46}$$

This means that up to $m = h$, the growth function increases linearly with the sample size. Thereafter, it only increases logarithmically, i.e., *much* more slowly. This is the regime where learning can succeed.

Although we do not make use of it in the present chapter, it is worthwhile to also introduce the *VC dimension of a class of real-valued functions* $\{f_{\mathbf{w}} | \mathbf{w} \in \Lambda\}$ at this stage. It is defined to equal the VC dimension of the class of indicator functions

$$\left\{ \operatorname{sgn}(f_{\mathbf{w}} - \beta) | \mathbf{w} \in \Lambda, \beta \in \left(\inf_x f_{\mathbf{w}}(x), \sup_x f_{\mathbf{w}}(x) \right) \right\}. \tag{5.47}$$

VC Dimension for Real-Valued Functions

In summary, we get a succession of capacity concepts,

$$H_{\mathcal{F}}(m) \leq H_{\mathcal{F}}^{\text{ann}}(m) \leq G_{\mathcal{F}}(m) \leq h \left(\ln \frac{m}{h} + 1 \right). \tag{5.48}$$

From left to right, these become less precise. The entropies on the left are distribution-dependent, but rather difficult to evaluate (see, e.g., [430, 391]). The growth function and VC dimension are distribution-independent. This is less accurate, and does not always capture the essence of a given problem, which might have a much more benign distribution than the worst case; on the other hand, we want the learning machine to work for all distributions. If we knew the distribution beforehand, then we would not need a learning machine anymore.

VC Dimension Example

Let us look at a simple example of the VC dimension. As a function class, we consider hyperplanes in $\mathbb{R}^2$, i.e.,

$$f(x) = \operatorname{sgn}(a + b[x]_1 + c[x]_2), \quad \text{with parameters } a, b, c \in \mathbb{R}. \tag{5.49}$$

Suppose we are given three points x_1, x_2, x_3 which are not collinear. No matter how they are labelled (that is, independent of our choice of $y_1, y_2, y_3 \in \{\pm 1\}$), we can always find parameters $a, b, c \in \mathbb{R}$ such that $f(x_i) = y_i$ for all i (see Figure 1.4 in the introduction). In other words, there exist three points that we can shatter. This

shows that the VC dimension of the set of hyperplanes in $\mathbb{R}^2$ satisfies $h \geq 3$. On the other hand, we can never shatter *four* points. It follows from simple geometry that given any four points, there is always a set of labels such that we cannot realize the corresponding classification. Therefore, the VC dimension is $h = 3$. More generally, for hyperplanes in $\mathbb{R}^N$, the VC dimension can be shown to be $h = N + 1$. For a formal derivation of this result, as well as of other examples, see [523].

VC Dimension of Hyperplanes

How does this fit together with the fact that SVMs can be shown to correspond to hyperplanes in feature spaces of possibly infinite dimension? The crucial point is that SVMs correspond to *large margin* hyperplanes. Once the margin enters, the capacity can be much smaller than the above general VC dimension of hyperplanes. For simplicity, we consider the case of hyperplanes containing the origin.

VC Dimension of Margin Hyperplanes

Theorem 5.5 (Vapnik [559]) *Consider hyperplanes* $\langle \mathbf{w}, \mathbf{x} \rangle = 0$, *where* $\mathbf{w}$ *is normalized such that they are in canonical form w.r.t. a set of points* $X^* = \{\mathbf{x}_1, \ldots, \mathbf{x}_r\}$; *i.e.,*

$$\min_{i=1,\ldots,r} | \langle \mathbf{w}, \mathbf{x}_i \rangle | = 1. \tag{5.50}$$

The set of decision functions $f_{\mathbf{w}}(\mathbf{x}) = \operatorname{sgn} \langle \mathbf{x}, \mathbf{w} \rangle$ *defined on* X^*, *and satisfying the constraint* $\|\mathbf{w}\| \leq \Lambda$, *has a VC dimension satisfying*

$$h \leq R^2 \Lambda^2. \tag{5.51}$$

Here, R is the radius of the smallest sphere centered at the origin and containing X^*.

Before we give a proof, several remarks are in order.

- The theorem states that we can control the VC dimension *irrespective of the dimension of the space* by controlling the length of the weight vector $\|\mathbf{w}\|$. Note, however, that this needs to be done a priori, by choosing a value for Λ. It therefore does not strictly motivate what we will later see in SVMs, where $\|\mathbf{w}\|$ is minimized in order to control the capacity. Detailed treatments can be found in the work of Shawe-Taylor et al. [491, 24, 125].
- There exists a similar result for the case where R is the radius of the smallest sphere (not necessarily centered at the origin) enclosing the data, and where we allow for the possibility that the hyperplanes have a nonzero offset b [562]. In this case, we give a simple visualization in figure Figure 5.4, which shows it is plausible that enforcing a large margin amounts to reducing the VC dimension.
- Note that the theorem talks about functions defined on X^*. To extend it to the case where the functions are defined on all of the input domain $\mathcal{X}$, it is best to state it for the *fat shattering dimension*. For details, see [24].

The proof [24, 222, 559] is somewhat technical, and can be skipped if desired.

Proof Let us assume that $\mathbf{x}_1, \ldots, \mathbf{x}_r$ are shattered by canonical hyperplanes with $\|\mathbf{w}\| \leq \Lambda$. Consequently, for all $y_1, \ldots, y_r \in \{\pm 1\}$, there exists a $\mathbf{w}$ with $\|\mathbf{w}\| \leq \Lambda$, such that

$$y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geq 1 \quad \text{for all } i = 1, \ldots, r. \tag{5.52}$$

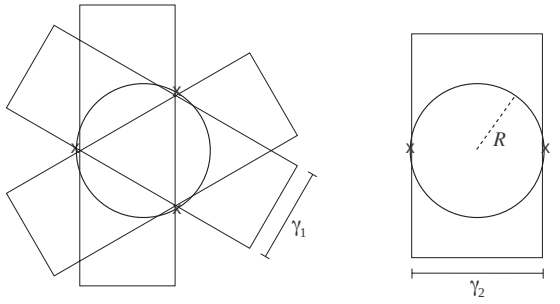


Figure 5.4 Simple visualization of the fact that enforcing a large margin of separation amounts to limiting the VC dimension. Assume that the data points are contained in a ball of radius R (cf. Theorem 5.5). Using hyperplanes with margin γ_1, it is possible to separate three points in all possible ways. Using hyperplanes with the larger margin γ_2, this is only possible for *two* points, hence the VC dimension in that case is two rather than three.

The proof proceeds in two steps. In the first part, we prove that the more points we want to shatter (5.52), the larger $\| \sum_{i=1}^r y_i \mathbf{x}_i \|$ must be. In the second part, we prove that we can upper bound the size of $\| \sum_{i=1}^r y_i \mathbf{x}_i \|$ in terms of R. Combining the two gives the desired condition, which tells us the maximum number of points we can shatter.

Summing (5.52) over $i = 1, \ldots, r$ yields

$$\left\langle \mathbf{w}, \left(\sum_{i=1}^r y_i \mathbf{x}_i \right) \right\rangle \geq r. \tag{5.53}$$

By the Cauchy-Schwarz inequality, on the other hand, we have

$$\left\langle \mathbf{w}, \left(\sum_{i=1}^r y_i \mathbf{x}_i \right) \right\rangle \leq \|\mathbf{w}\| \left\| \sum_{i=1}^r y_i \mathbf{x}_i \right\| \leq \Lambda \left\| \sum_{i=1}^r y_i \mathbf{x}_i \right\|. \tag{5.54}$$

Here, the second inequality follows from $\|\mathbf{w}\| \leq \Lambda$.

Combining (5.53) and (5.54), we get the desired lower bound,

$$\frac{r}{\Lambda} \leq \left\| \sum_{i=1}^r y_i \mathbf{x}_i \right\|. \tag{5.55}$$

We now move on to the second part. Let us consider independent random labels $y_i \in \{\pm 1\}$ which are uniformly distributed, sometimes called *Rademacher variables*. Let $\mathbf{E}$ denote the expectation over the choice of the labels. Exploiting the linearity of $\mathbf{E}$, we have

$$\begin{aligned} \mathbf{E} \left\| \sum_{i=1}^r y_i \mathbf{x}_i \right\|^2 &= \sum_{i=1}^r \mathbf{E} \left\langle y_i \mathbf{x}_i, \sum_{j=1}^r y_j \mathbf{x}_j \right\rangle \\ &= \sum_{i=1}^r \mathbf{E} \left\langle y_i \mathbf{x}_i, \left(\left(\sum_{j \neq i} y_j \mathbf{x}_j \right) + y_i \mathbf{x}_i \right) \right\rangle \end{aligned}$$

$$= \sum_{i=1}^{r} \left(\left(\sum_{j \neq i} \mathbf{E} \left\langle y_i \mathbf{x}_i, y_j \mathbf{x}_j \right\rangle \right) + \mathbf{E} \left\langle y_i \mathbf{x}_i, y_i \mathbf{x}_i \right\rangle \right)$$

$$= \sum_{i=1}^{r} \mathbf{E} \|y_i \mathbf{x}_i\|^2, \tag{5.56}$$

where the last equality follows from the fact that the Rademacher variables have zero mean and are independent. Exploiting the fact that $\|y_i \mathbf{x}_i\| = \|\mathbf{x}_i\| \leq R$, we get

$$\mathbf{E} \left\| \sum_{i=1}^{r} y_i \mathbf{x}_i \right\|^2 \leq rR^2. \tag{5.57}$$

Since this is true for the expectation over the random choice of the labels, there must be at least one set of labels for which it also holds true. We have so far made no restrictions on the labels, hence we may now use this specific set of labels. This leads to the desired upper bound,

$$\left\| \sum_{i=1}^{r} y_i \mathbf{x}_i \right\|^2 \leq rR^2. \tag{5.58}$$

Combining the upper bound with the lower bound (5.55), we get

$$\frac{r^2}{\Lambda^2} \leq rR^2; \tag{5.59}$$

hence,

$$r \leq R^2 \Lambda^2. \tag{5.60}$$

In other words, if the r points are shattered by a canonical hyperplane satisfying the assumptions we have made, then r is constrained by (5.60). The VC dimension h also satisfies (5.60), since it corresponds to the maximum number of points that can be shattered. ■

In the next section, we give an application of this theorem. Readers only interested in the theoretical background of learning theory may want to skip this section.

5.6 A Model Selection Example

In the following example, taken from [470], we use a bound of the form (5.36) to predict which kernel would perform best on a character recognition problem (USPS set, see Section A.1). Since the problem is essentially separable, we disregard the empirical risk term in the bound, and choose the parameters of a polynomial kernel by minimizing the second term. Note that the second term is a monotonic function of the capacity. As a capacity measure, we use the upper bound on the VC dimension described in Theorem 5.5, which in turn is an upper bound on the logarithm of the covering number that appears in (5.36) (by the arguments put forward in Section 5.5.6).

$$= \sum_{i=1}^{r} \left(\left(\sum_{j \neq i} \mathbf{E} \left\langle y_i \mathbf{x}_i, y_j \mathbf{x}_j \right\rangle \right) + \mathbf{E} \left\langle y_i \mathbf{x}_i, y_i \mathbf{x}_i \right\rangle \right)$$

$$= \sum_{i=1}^{r} \mathbf{E} \|y_i \mathbf{x}_i\|^2, \tag{5.56}$$

where the last equality follows from the fact that the Rademacher variables have zero mean and are independent. Exploiting the fact that $\|y_i \mathbf{x}_i\| = \|\mathbf{x}_i\| \leq R$, we get

$$\mathbf{E} \left\| \sum_{i=1}^{r} y_i \mathbf{x}_i \right\|^2 \leq rR^2. \tag{5.57}$$

Since this is true for the expectation over the random choice of the labels, there must be at least one set of labels for which it also holds true. We have so far made no restrictions on the labels, hence we may now use this specific set of labels. This leads to the desired upper bound,

$$\left\| \sum_{i=1}^{r} y_i \mathbf{x}_i \right\|^2 \leq rR^2. \tag{5.58}$$

Combining the upper bound with the lower bound (5.55), we get

$$\frac{r^2}{\Lambda^2} \leq rR^2; \tag{5.59}$$

hence,

$$r \leq R^2 \Lambda^2. \tag{5.60}$$

In other words, if the r points are shattered by a canonical hyperplane satisfying the assumptions we have made, then r is constrained by (5.60). The VC dimension h also satisfies (5.60), since it corresponds to the maximum number of points that can be shattered. ■

In the next section, we give an application of this theorem. Readers only interested in the theoretical background of learning theory may want to skip this section.

5.6 A Model Selection Example

In the following example, taken from [470], we use a bound of the form (5.36) to predict which kernel would perform best on a character recognition problem (USPS set, see Section A.1). Since the problem is essentially separable, we disregard the empirical risk term in the bound, and choose the parameters of a polynomial kernel by minimizing the second term. Note that the second term is a monotonic function of the capacity. As a capacity measure, we use the upper bound on the VC dimension described in Theorem 5.5, which in turn is an upper bound on the logarithm of the covering number that appears in (5.36) (by the arguments put forward in Section 5.5.6).

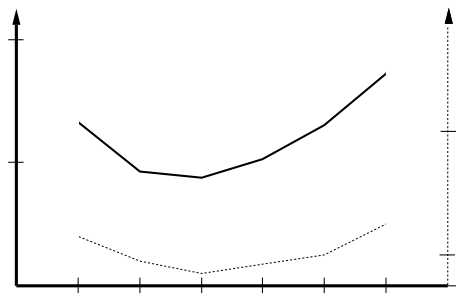


Figure 5.5 Average VC dimension (solid), and total number of test errors, of ten two-class-classifiers (dotted) with polynomial degrees 2 through 7, trained on the USPS set of handwritten digits. The baseline 174 on the error scale, corresponds to the total number of test errors of the ten *best* binary classifiers, chosen from degrees 2 through 7. The graph shows that for this problem, which can essentially be solved with zero training error for all degrees greater than 1, the VC dimension allows us to predict that degree 4 yields the best overall performance of the two-class-classifier on the test set (from [470, 467]).

Computing the Enclosing Sphere in $\mathcal{H}$

We employ a version of Theorem 5.5, which uses the radius of the smallest sphere containing the data in a feature space $\mathcal{H}$ associated with the kernel k [561]. The radius was computed by solving a quadratic program [470, 85] (cf. Section 8.3). We formulate the problem as follows:

$$\begin{array}{ll} \underset{R\geq 0,\mathbf{x}^*\in\mathcal{H}}{\text{minimize}} & R^2 \\ \text{subject to} & \|\mathbf{x}_i-\mathbf{x}^*\|^2 \leq R^2, \end{array} \tag{5.61}$$

where $\mathbf{x}^*$ is the center of the sphere, and is found in the course of the optimization. Employing the tools of constrained optimization, as briefly described in Chapter 1 (for details, see Chapter 6), we construct a Lagrangian,

$$R^2 - \sum_{i=1}^{m} \lambda_i (R^2 - (\mathbf{x}_i - \mathbf{x}^*)^2), \tag{5.62}$$

and compute the derivatives with respect to $\mathbf{x}^*$ and R, to get

$$\mathbf{x}^* = \sum_{i=1}^{m} \lambda_i \mathbf{x}_i, \tag{5.63}$$

and the Wolfe dual problem:

$$\underset{\boldsymbol{\lambda}\in\mathbb{R}^m}{\text{maximize}} \quad \sum_{i=1}^{m} \lambda_i \cdot \langle \mathbf{x}_i, \mathbf{x}_i \rangle - \sum_{i,j=1}^{m} \lambda_i \lambda_j \cdot \langle \mathbf{x}_i, \mathbf{x}_j \rangle, \tag{5.64}$$

$$\text{subject to} \quad \sum_{i=1}^{m} \lambda_i = 1, \ \lambda_i \geq 0, \tag{5.65}$$

where $\boldsymbol{\lambda}$ is the vector of all Lagrange multipliers $\lambda_i, i = 1, \ldots, m$.

As in the Support Vector algorithm, this problem has the property that the $\mathbf{x}_i$

appear only in dot products, so we can again compute the dot products in feature space, replacing $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ by $k(x_i, x_j)$ (where the x_i belong to the input domain $\mathcal{X}$, and the $\mathbf{x}_i$ in the feature space $\mathcal{H}$).

As Figure 5.5 shows, the VC dimension bound, using the radius R computed in this way, gives a rather good prediction of the error on an independent test set.

5.7 Summary

In this chapter, we introduced the main ideas of statistical learning theory. For learning processes utilizing empirical risk minimization to be successful, we need a version of the law of large numbers that holds uniformly over all functions the learning machine can implement. For this uniform law to hold true, the capacity of the set of functions that the learning machine can implement has to be "well-behaved." We gave several capacity measures, such as the VC dimension, and illustrated how to derive bounds on the test error of a learning machine, in terms of the training error and the capacity. We have, moreover, shown how to bound the capacity of margin classifiers, a result which will later be used to motivate the Support Vector algorithm. Finally, we described an application in which a uniform convergence bound was used for model selection.

Whilst this discussion of learning theory should be sufficient to understand most of the present book, we will revisit learning theory at a later stage. In Chapter 12, we will present some more advanced material, which applies to kernel learning machines. Specifically, we will introduce another class of generalization error bound, building on a concept of *stability* of algorithms minimizing regularized risk functionals. These bounds are proven using concentration-of-measure inequalities, which are themselves generalizations of Chernoff and Hoeffding type bounds. In addition, we will discuss *leave-one-out* and *PAC-Bayesian* bounds.

5.8 Problems

5.1 (No Free Lunch in Kernel Choice ••) *Discuss the relationship between the "no-free-lunch Theorem" and the statement that there is no free lunch in kernel choice.*

5.2 (Error Counting Estimate [136] •) *Suppose you are given a test set with n elements to assess the accuracy of a trained classifier. Use the Chernoff bound to quantify the probability that the mean error on the test set differs from the true risk by more than $\epsilon > 0$. Argue that the test set should be as large as possible, in order to get a reliable estimate of the performance of a classifier.*

5.3 (The Tainted Die ••) *A con-artist wants to taint a die such that it does not generate any '6' when cast. Yet he does not know exactly how. So he devises the following scheme:*

appear only in dot products, so we can again compute the dot products in feature space, replacing $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ by $k(x_i, x_j)$ (where the x_i belong to the input domain $\mathcal{X}$, and the $\mathbf{x}_i$ in the feature space $\mathcal{H}$).

As Figure 5.5 shows, the VC dimension bound, using the radius R computed in this way, gives a rather good prediction of the error on an independent test set.

5.7 Summary

In this chapter, we introduced the main ideas of statistical learning theory. For learning processes utilizing empirical risk minimization to be successful, we need a version of the law of large numbers that holds uniformly over all functions the learning machine can implement. For this uniform law to hold true, the capacity of the set of functions that the learning machine can implement has to be "well-behaved." We gave several capacity measures, such as the VC dimension, and illustrated how to derive bounds on the test error of a learning machine, in terms of the training error and the capacity. We have, moreover, shown how to bound the capacity of margin classifiers, a result which will later be used to motivate the Support Vector algorithm. Finally, we described an application in which a uniform convergence bound was used for model selection.

Whilst this discussion of learning theory should be sufficient to understand most of the present book, we will revisit learning theory at a later stage. In Chapter 12, we will present some more advanced material, which applies to kernel learning machines. Specifically, we will introduce another class of generalization error bound, building on a concept of *stability* of algorithms minimizing regularized risk functionals. These bounds are proven using concentration-of-measure inequalities, which are themselves generalizations of Chernoff and Hoeffding type bounds. In addition, we will discuss *leave-one-out* and *PAC-Bayesian* bounds.

5.8 Problems

5.1 (No Free Lunch in Kernel Choice ••) *Discuss the relationship between the "no-free-lunch Theorem" and the statement that there is no free lunch in kernel choice.*

5.2 (Error Counting Estimate [136] •) *Suppose you are given a test set with n elements to assess the accuracy of a trained classifier. Use the Chernoff bound to quantify the probability that the mean error on the test set differs from the true risk by more than $\epsilon > 0$. Argue that the test set should be as large as possible, in order to get a reliable estimate of the performance of a classifier.*

5.3 (The Tainted Die ••) *A con-artist wants to taint a die such that it does not generate any '6' when cast. Yet he does not know exactly how. So he devises the following scheme:*

appear only in dot products, so we can again compute the dot products in feature space, replacing $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ by $k(x_i, x_j)$ (where the x_i belong to the input domain $\mathcal{X}$, and the $\mathbf{x}_i$ in the feature space $\mathcal{H}$).

As Figure 5.5 shows, the VC dimension bound, using the radius R computed in this way, gives a rather good prediction of the error on an independent test set.

5.7 Summary

In this chapter, we introduced the main ideas of statistical learning theory. For learning processes utilizing empirical risk minimization to be successful, we need a version of the law of large numbers that holds uniformly over all functions the learning machine can implement. For this uniform law to hold true, the capacity of the set of functions that the learning machine can implement has to be "well-behaved." We gave several capacity measures, such as the VC dimension, and illustrated how to derive bounds on the test error of a learning machine, in terms of the training error and the capacity. We have, moreover, shown how to bound the capacity of margin classifiers, a result which will later be used to motivate the Support Vector algorithm. Finally, we described an application in which a uniform convergence bound was used for model selection.

Whilst this discussion of learning theory should be sufficient to understand most of the present book, we will revisit learning theory at a later stage. In Chapter 12, we will present some more advanced material, which applies to kernel learning machines. Specifically, we will introduce another class of generalization error bound, building on a concept of *stability* of algorithms minimizing regularized risk functionals. These bounds are proven using concentration-of-measure inequalities, which are themselves generalizations of Chernoff and Hoeffding type bounds. In addition, we will discuss *leave-one-out* and *PAC-Bayesian* bounds.

5.8 Problems

5.1 (No Free Lunch in Kernel Choice ••) *Discuss the relationship between the "no-free-lunch Theorem" and the statement that there is no free lunch in kernel choice.*

5.2 (Error Counting Estimate [136] •) *Suppose you are given a test set with n elements to assess the accuracy of a trained classifier. Use the Chernoff bound to quantify the probability that the mean error on the test set differs from the true risk by more than $\epsilon > 0$. Argue that the test set should be as large as possible, in order to get a reliable estimate of the performance of a classifier.*

5.3 (The Tainted Die ••) *A con-artist wants to taint a die such that it does not generate any '6' when cast. Yet he does not know exactly how. So he devises the following scheme:*

he makes some changes and subsequently rolls the die 20 times to check that no '6' occurs. Unless pleased with the outcome, he changes more things and repeats the experiment.

How long will it take on average, until, even with a perfect die, he will be convinced that he has a die that never generates a '6'? What is the probability that this already happens at the first trial? Can you improve the strategy such that he can be sure the die is 'well' tainted (hint: longer trials provide increased confidence)?

5.4 (Chernoff Bound for the Deviation of Empirical Means ••) *Use (5.6) and the triangle inequality to prove that*

$$\mathrm{P}\left\{\left|\frac{1}{m}\sum_{i=1}^{m}\xi_i - \frac{1}{m}\sum_{i=m+1}^{2m}\xi_i\right| \geq \epsilon\right\} \leq 4\,\exp\left(-\frac{m\epsilon^2}{2}\right). \tag{5.66}$$

Next, note that the bound (5.66) is symmetric in how it deals with the two halves of the sample. Therefore, since the two events

$$\left\{\frac{1}{m}\sum_{i=1}^{m}\xi_i - \frac{1}{m}\sum_{i=m+1}^{2m}\xi_i \geq \epsilon\right\} \tag{5.67}$$

and

$$\left\{\frac{1}{m}\sum_{i=1}^{m}\xi_i - \frac{1}{m}\sum_{i=m+1}^{2m}\xi_i \leq -\epsilon\right\} \tag{5.68}$$

are disjoint, argue that (5.32) holds true. See also Corollary 6.34 below.

5.5 (Consistency and Uniform Convergence ••) *Why can we not get a bound on the generalization error of a learning algorithm by applying (5.11) to the outcome of the algorithm? Argue that since we do not know in advance which function the learning algorithm returns, we need to consider the worst possible case, which leads to uniform convergence considerations.*

Speculate whether there could be restrictions on learning algorithms which imply that effectively, *empirical risk minimization only leads to a subset of the set of all possible functions. Argue that this amounts to restricting the capacity. Consider as an example neural networks with back-propagation: if the training algorithm always returns a local minimum close to the starting point in weight space, then the network effectively does not explore the whole weight (i.e., function) space.*

5.6 (Confidence Interval and Uniform Convergence •) *Derive (5.36) from (5.35).*

5.7 (Representer Algorithms for Minimizing VC Bounds ○○○) *Construct kernel algorithms that are more closely aligned with VC bounds of the form (5.36). Hint: in the risk functional, replace the standard SV regularizer $\|\mathbf{w}\|^2$ with the second term of (5.36), bounding the shattering coefficient with the VC dimension bound (Theorem 5.5). Use the representer theorem (Section 4.2) to argue that the minimizer takes the form of a kernel expansion in terms of the training examples. Find the optimal expansion coefficients by minimizing the modified risk functional over the choice of expansion coefficients.*

5.8 (Bounds in Terms of the VC Dimension •) *From (5.35) and (5.36), derive bounds in terms of the growth function and the VC dimension, using the results of Section 5.5.6. Discuss the conditions under which they hold.*

5.9 (VC Theory and Decision Theory •••) *(i) Discuss the relationship between minimax estimation (cf. footnote 7 in Chapter 1) and VC theory. Argue that the VC bounds can be made "worst case" over distributions by picking suitable capacity measures. However, they only bound the difference between empirical risk and true risk, thus they are only "worst case" for the variance term, not for the bias (or empirical risk). The minimization of an upper bound on the risk of the form (5.36) as performed in SRM is done in order to construct an induction principle rather than to make a minimax statement. Finally, note that the minimization is done with respect to a structure on the set of functions, while in the minimax paradigm one takes the minimum directly over (all) functions.*

(ii) Discuss the following folklore statement: "VC statisticians do not care about doing the optimal thing, as long as they can guarantee how well they are doing. Bayesians do not care how well they are doing, as long as they are doing the optimal thing."

5.10 (Overfitting on the Test Set •••) *Consider a learning algorithm which has a free parameter C. Suppose you randomly pick n values $C_1, \ldots, C_n$, and for each n, you train your algorithm. At the end, you pick the value for C which did best on the test set. How would you expect your misjudgment of the true test error to scale with n?*

How does the situation change if the C_i are not picked randomly, but by some adaptive scheme which proposes new values of C by looking at how the previous ones did, and guessing which change of C would likely improve the performance on the test set?

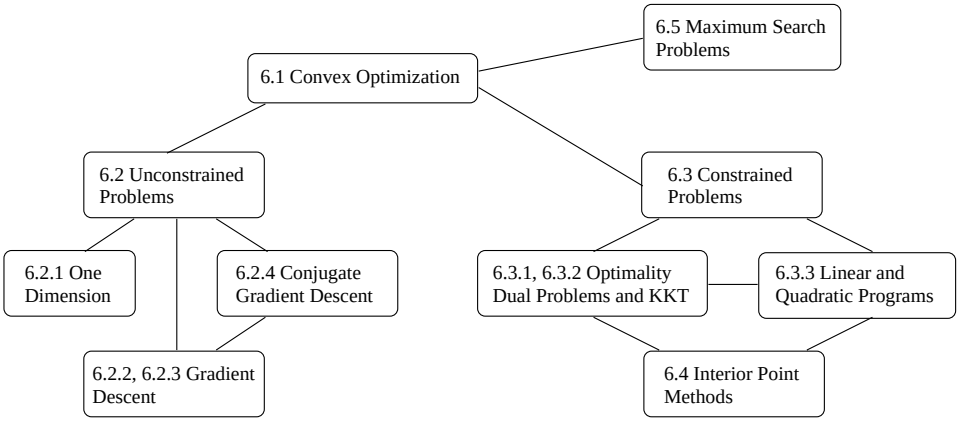
5.11 (Overfitting the Leave-One-Out Error ••) *Explain how it is possible to overfit the leave-one-out error. I.e., consider a learning algorithm that minimizes the leave-one-out error, and argue that it is possible that this algorithm will overfit.*

5.12 (Learning Theory for Differential Equations ○○○) *Can you develop a statistical theory of estimating differential equations from data? How can one suitably restrict the "capacity" of differential equations?*

Note that without restrictions, already ordinary differential equations may exhibit behavior where the capacity is infinite, as exemplified by Rubel's universal differential equation [447]

$$\begin{aligned} 3y'^4y''y'''^2 - 4y'^4y'''^2y'''' + 6y'^3y''^2y'''y'''' + 24y'^2y''^4y'''' \\ -12y'^3y''y'''^3 - 29y'^2y''^3y'''^2 + 12y''^7 \quad = 0. \end{aligned} \tag{5.69}$$

Rubel proved that given any continuous function $f : \mathbb{R} \to \mathbb{R}$ and any positive continuous function $\varepsilon : \mathbb{R} \to \mathbb{R}^+$, there exists a C^∞ solution y of (5.69) such that $|y(t) - f(t)| < \varepsilon(t)$ for all $t \in \mathbb{R}$. Therefore, all continuous functions are uniform limits of sequences of solutions of (5.69). Moreover, y can be made to agree with f at a countable number of distinct points (t_i). Further references of interest to this problem include [61, 78, 63].



optimization problems (Section 6.2) are less common in this book and will only be required in the gradient descent methods of Section 10.6.1, and the Gaussian Process implementation methods of Section 16.4.

Prerequisites

The present chapter is intended as an introduction to the basic concepts of optimization. It is relatively self-contained, and requires only basic skills in linear algebra and multivariate calculus. Section 6.3 is somewhat more technical, Section 6.4 requires some additional knowledge of numerical analysis, and Section 6.5 assumes some knowledge of probability and statistics.

6.1 Convex Optimization

In the situations considered in this book, learning (or equivalently statistical estimation) implies the minimization of some risk functional such as $R_{\text{emp}}[f]$ or $R_{\text{reg}}[f]$ (cf. Chapter 4). While minimizing an arbitrary function on a (possibly not even compact) set of arguments can be a difficult task, and will most likely exhibit many local minima, minimization of a convex objective function on a convex set exhibits exactly one *global* minimum. We now prove this property.

Definition and Construction of Convex Sets and Functions

Definition 6.1 (Convex Set) *A set X in a vector space is called convex if for any $x, x' \in X$ and any $\lambda \in [0,1]$, we have*

$$\lambda x + (1-\lambda)x' \in X. \tag{6.1}$$

Definition 6.2 (Convex Function) *A function f defined on a set X (note that X need not be convex itself) is called convex if, for any $x, x' \in X$ and any $\lambda \in [0,1]$ such that $\lambda x + (1-\lambda)x' \in X$, we have*

$$f(\lambda x + (1-\lambda)x') \leq \lambda f(x) + (1-\lambda) f(x'). \tag{6.2}$$

A function f is called strictly *convex if for $x \neq x'$ and $\lambda \in (0,1)$ (6.2) is a strict inequality.*

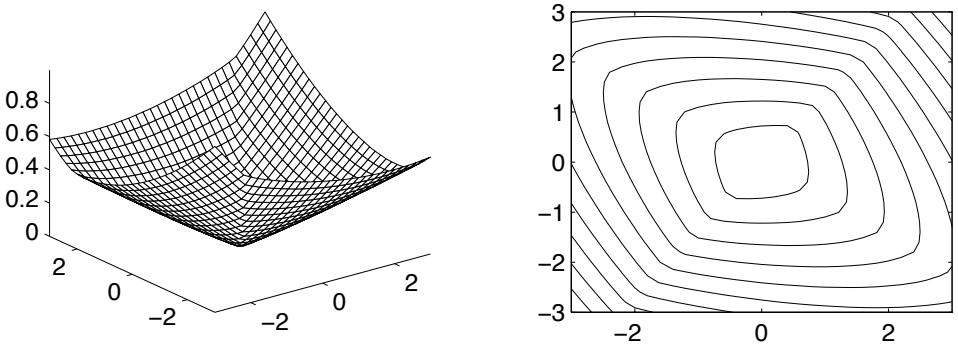


Figure 6.1 Left: Convex Function in two variables. Right: the corresponding convex level sets $\{x|f(x) \leq c\}$, for different values of c.

There exist several ways to define convex sets. A convenient method is to define them via *below sets* of convex functions, such as the sets for which $f(x) \leq c$, for instance.

Lemma 6.3 (Convex Sets as Below-Sets) *Denote by $f : \mathcal{X} \to \mathbb{R}$ a convex function on a convex set $\mathcal{X}$. Then the set*

$$X := \{x|x \in \mathcal{X} \text{ and } f(x) \leq c\}, \text{ for all } c \in \mathbb{R}, \tag{6.3}$$

is convex.

Proof We must show condition (6.1). For any $x, x' \in \mathcal{X}$, we have $f(x), f(x') \leq c$. Moreover, since f is convex, we also have

$$f(\lambda x + (1-\lambda)x') \leq \lambda f(x) + (1-\lambda)f(x') \leq c \text{ for all } \lambda \in [0,1]. \tag{6.4}$$

Hence, for all $\lambda \in [0,1]$, we have $(\lambda x + (1-\lambda)x') \in X$, which proves the claim. Figure 6.1 depicts this situation graphically. ■

Lemma 6.4 (Intersection of Convex Sets) *Denote by $X, X' \subset \mathcal{X}$ two convex sets. Then $X \cap X'$ is also a convex set.*

Intersections

Proof Given any $x, x' \in X \cap X'$, then for any $\lambda \in [0,1]$, the point $x_\lambda := \lambda x + (1-\lambda)x'$ satisfies $x_\lambda \in X$ and $x_\lambda \in X'$, hence also $x_\lambda \in X \cap X'$. ■

See also Figure 6.2. Now we have the tools to prove the central theorem of this section.

Theorem 6.5 (Minima on Convex Sets) *If the convex function $f : \mathcal{X} \to \mathbb{R}$ has a minimum on a convex set $X \subset \mathcal{X}$, then its arguments $x \in \mathcal{X}$, for which the minimum value is attained, form a convex set. Moreover, if f is strictly convex, then this set will contain only one element.*

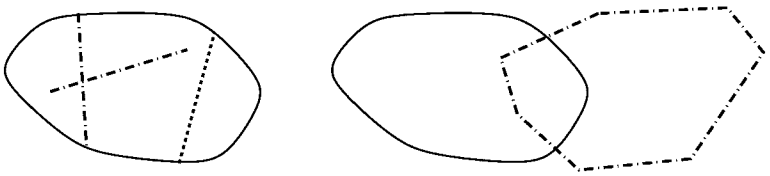


Figure 6.2 Left: a convex set; observe that lines with points in the set are fully contained inside the set. Right: the intersection of two convex sets is also a convex set.

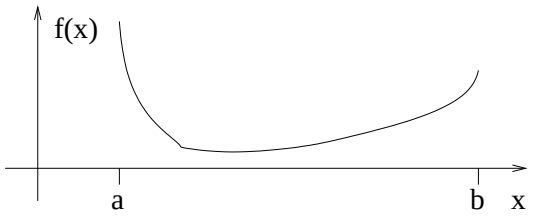


Figure 6.3 Note that the maximum of a convex function is obtained at the ends of the interval $[a, b]$.

Proof Denote by c the minimum of f on X. Then the set $X_m := \{x | x \in \mathcal{X} \text{ and } f(x) \leq c\}$ is clearly convex. In addition, $X_m \cap X$ is also convex, and $f(x) = c$ for all $x \in X_m \cap X$ (otherwise c would not be the minimum).

If f is strictly convex, then for any $x, x' \in X$, and in particular for any $x, x' \in X \cap X_m$, we have (for $x \neq x'$ and all $\lambda \in (0, 1)$),

$$f(\lambda x + (1 - \lambda)x') < \lambda f(x) + (1 - \lambda) f(x') = \lambda c + (1 - \lambda)c = c. \tag{6.5}$$

This contradicts the assumption that $X_m \cap X$ contains more then one element.

■

Global Minima

A simple application of this theorem is in constrained convex minimization. Recall that the notation $[n]$, used below, is a shorthand for $\{1, \ldots, n\}$.

Corollary 6.6 (Constrained Convex Minimization) *Given the set of convex functions $f, c_1, \ldots, c_n$ on the convex set $\mathcal{X}$, the problem*

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f(x), \\ \textit{subject to} & c_i(x) \leq 0 \textit{ for all } i \in [n], \end{array} \tag{6.6}$$

has as its solution a convex set, if a solution exists. This solution is unique if f is strictly convex.

Many problems in Mathematical Programming or Support Vector Machines can be cast into this formulation. This means either that they all have unique solutions (if f is strictly convex), or that all solutions are equally good and form a convex set (if f is merely convex).

We might ask what can be said about convex *maximization*. Let us analyze a simple case first: convex maximization on an interval.

Lemma 6.7 (Convex Maximization on an Interval) *Denote by f a convex function on $[a, b] \in \mathbb{R}$. Then the problem of maximizing f on $[a, b]$ has $f(a)$ and $f(b)$ as solutions.*

Maxima on Extreme Points

Proof Any $x \in [a, b]$ can be written as $\frac{b-x}{b-a}a + \left(1 - \frac{b-x}{b-a}\right) b$, and hence

$$f(x) \leq \frac{b-x}{b-a} f(a) + \left(1 - \frac{b-x}{b-a}\right) f(b) \leq \max(f(a), f(b)). \tag{6.7}$$

Therefore the maximum of f on $[a, b]$ is obtained on one of the points a, b. ∎

We will next show that the problem of convex *maximization* on a convex set is typically a hard problem, in the sense that the maximum can only be found at one of the extreme points of the constraining set. We must first introduce the notion of vertices of a set.

Definition 6.8 (Vertex of a Set) *A point $x \in X$ is a vertex of X if, for all $x' \in X$ with $x' \neq x$, and for all $\lambda > 1$, the point $\lambda x + (1 - \lambda)x' \notin X$.*

This definition implies, for instance, that in the case of X being an ℓ_2 ball, the vertices of X make up its surface. In the case of an ℓ_∞ ball, we have 2^n vertices in n dimensions, and for an ℓ_1 ball, we have only $2n$ of them. These differences will guide us in the choice of admissible sets of parameters for optimization problems (see, e.g., Section 14.4). In particular, there exists a connection between suprema on sets and their convex hulls. To state this link, however, we need to define the latter.

Definition 6.9 (Convex Hull) *Denote by X a set in a vector space. Then the convex hull* $\operatorname{co} X$ *is defined as*

$$\operatorname{co} X := \left\{ \bar{x} \,\middle|\, \bar{x} = \sum_{i=1}^{n} \alpha_i x_i \text{ where } n \in \mathbb{N}, \alpha_i \geq 0 \text{ and } \sum_{i=1}^{n} \alpha_i = 1 \right\}. \tag{6.8}$$

Theorem 6.10 (Suprema on Sets and their Convex Hulls) *Denote by X a set and by* $\operatorname{co} X$ *its convex hull. Then for a convex function f*

$$\sup\{f(x) | x \in X\} = \sup\{f(x) | x \in \operatorname{co} X\}. \tag{6.9}$$

Evaluating Convex Sets on Extreme Points

Proof Recall that the below set of convex functions is convex (Lemma 6.3), and that the below set of f with respect to $c = \sup\{f(x) | x \in X\}$ is by definition a superset of X. Moreover, due to its convexity, it is also a superset of $\operatorname{co} X$. ∎

This theorem can be used to replace search operations over sets X by subsets $X' \subset X$, which are considerably smaller, if the convex hull of the latter generates X. In particular, the vertices of convex sets are sufficient to reconstruct the whole set.

Theorem 6.11 (Vertices) *A compact convex set is the convex hull of its vertices.*

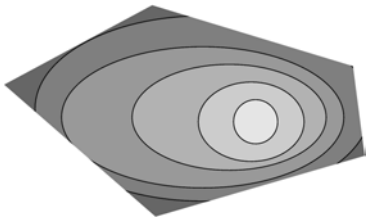


Figure 6.4 A convex function on a convex polyhedral set. Note that the minimum of this function is unique, and that the maximum can be found at one of the vertices of the constraining domain.

Reconstructing Convex Sets from Vertices

The proof is slightly technical, and not central to the understanding of kernel methods. See Rockafellar [435, Chapter 18] for details, along with further theorems on convex functions. We now proceed to the second key theorem in this section.

Theorem 6.12 (Maxima of Convex Functions on Convex Compact Sets) *Denote by X a compact convex set in $\mathcal{X}$, by $|X$ the vertices of X, and by f a convex function on X. Then*

$$\sup\{f(x)|x \in X\} = \sup\{f(x)|x \in |X\}. \tag{6.10}$$

Proof Application of Theorem 6.10 and Theorem 6.11 proves the claim, since under the assumptions made on X, we have $X = \mathrm{co}(|X)$. Figure 6.4 depicts the situation graphically. ■

6.2 Unconstrained Problems

After the characterization and uniqueness results (Theorem 6.5, Corollary 6.6, and Lemma 6.7) of the previous section, we will now study numerical techniques to obtain minima (or maxima) of convex optimization problems. While the choice of algorithms is motivated by applicability to kernel methods, the presentation here is not problem specific. For details on implementation, and descriptions of applications to learning problems, see Chapter 10.

6.2.1 Functions of One Variable

We begin with the easiest case, in which f depends on only one variable. Some of the concepts explained here, such as the interval cutting algorithm and Newton's method, can be extended to the multivariate setting (see Problem 6.5). For the sake of simplicity, however, we limit ourselves to the univariate case.

Assume we want to minimize $f : \mathbb{R} \to \mathbb{R}$ on the interval $[a, b] \subset \mathbb{R}$. If we cannot make any further assumptions regarding f, then this problem, as simple as it may seem, cannot be solved numerically.

Continuous Differentiable Functions

If f is *differentiable*, the problem can be reduced to finding $f'(x) = 0$ (see Problem 6.4 for the general case). If in addition to the previous assumptions, f is convex, then f' is nondecreasing, and we can find a fast, simple algorithm (Algorithm

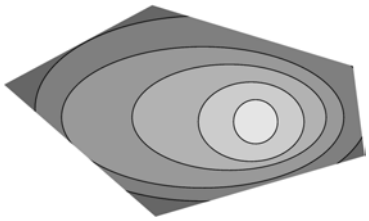


Figure 6.4 A convex function on a convex polyhedral set. Note that the minimum of this function is unique, and that the maximum can be found at one of the vertices of the constraining domain.

Reconstructing Convex Sets from Vertices

The proof is slightly technical, and not central to the understanding of kernel methods. See Rockafellar [435, Chapter 18] for details, along with further theorems on convex functions. We now proceed to the second key theorem in this section.

Theorem 6.12 (Maxima of Convex Functions on Convex Compact Sets) *Denote by X a compact convex set in $\mathcal{X}$, by $|X$ the vertices of X, and by f a convex function on X. Then*

$$\sup\{f(x)|x \in X\} = \sup\{f(x)|x \in |X\}. \tag{6.10}$$

Proof Application of Theorem 6.10 and Theorem 6.11 proves the claim, since under the assumptions made on X, we have $X = \mathrm{co}(|X)$. Figure 6.4 depicts the situation graphically. ■

6.2 Unconstrained Problems

After the characterization and uniqueness results (Theorem 6.5, Corollary 6.6, and Lemma 6.7) of the previous section, we will now study numerical techniques to obtain minima (or maxima) of convex optimization problems. While the choice of algorithms is motivated by applicability to kernel methods, the presentation here is not problem specific. For details on implementation, and descriptions of applications to learning problems, see Chapter 10.

6.2.1 Functions of One Variable

We begin with the easiest case, in which f depends on only one variable. Some of the concepts explained here, such as the interval cutting algorithm and Newton's method, can be extended to the multivariate setting (see Problem 6.5). For the sake of simplicity, however, we limit ourselves to the univariate case.

Assume we want to minimize $f : \mathbb{R} \to \mathbb{R}$ on the interval $[a, b] \subset \mathbb{R}$. If we cannot make any further assumptions regarding f, then this problem, as simple as it may seem, cannot be solved numerically.

Continuous Differentiable Functions

If f is *differentiable*, the problem can be reduced to finding $f'(x) = 0$ (see Problem 6.4 for the general case). If in addition to the previous assumptions, f is convex, then f' is nondecreasing, and we can find a fast, simple algorithm (Algorithm

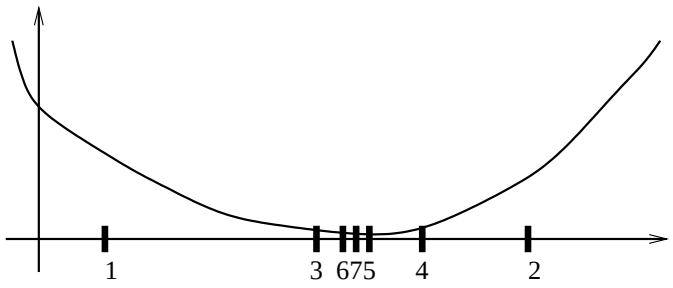


Figure 6.5 Interval Cutting Algorithm. The selection of points is ordered according to the numbers beneath (points 1 and 2 are the initial endpoints of the interval).

Algorithm 6.1 Interval Cutting

Require: a, b, Precision ϵ
  Set $A = a, B = b$
  repeat
    if $f'\left(\frac{A+B}{2}\right) > 0$ **then**
      $B = \frac{A+B}{2}$
    else
      $A = \frac{A+B}{2}$
    end if
  until $(B - A)\min(|f'(A)|, |f'(B)|) \leq \epsilon$
Output: $x = \frac{A+B}{2}$

Interval Cutting

6.1) to solve our problem (see Figure 6.5).

This technique works by halving the size of the interval that contains the minimum x^* of f, since it is always guaranteed by the selection criteria for B and A that $x^* \in [A, B]$. We use the following Taylor series expansion to determine the stopping criterion.

Theorem 6.13 (Taylor Series) *Denote by $f : \mathbb{R} \to \mathbb{R}$ a function that is d times differentiable. Then for any $x, x' \in \mathbb{R}$, there exists a ξ with $|\xi| \leq |x - x'|$, such that*

$$f(x') = \sum_{i=0}^{d-1} \frac{1}{i!} f^{(i)}(x)(x' - x)^i + \frac{\xi^d}{d!} f^{(d)}(x + \xi). \tag{6.11}$$

Now we may apply (6.11) to the stopping criterion of Algorithm 6.1. We denote by x^* the minimum of $f(x)$. Expanding f around $f(x^*)$, we obtain for some $\xi_A \in [A - x^*, 0]$ that $f(A) = f(x^*) + \xi_A f'(x^* + \xi_A)$, and therefore,

$$|f(A) - f(x^*)| = |\xi_A||f'(x^* + \xi_A)| \leq (B - A)|f'(A)|.$$

Proof of Linear Convergence

Taking the minimum over $\{A, B\}$ shows that Algorithm 6.1 stops once f is ϵ-close to its minimal value. The convergence of the algorithm is *linear* with constant 0.5, since the intervals $[A, B]$ for possible x^* are halved at each iteration.

Algorithm 6.2 Newton's Method

Require: x_0, Precision ϵ
  Set $x = x_0$
  repeat
    $x = x - \frac{f'(x)}{f''(x)}$
  until $|f'(x)| \leq \epsilon$
Output: x

In constructing the interval cutting algorithm, we in fact wasted most of the information obtained in evaluating f' at each point, by only making use of the sign of f. In particular, we could fit a parabola to f and thereby obtain a method that converges more rapidly. If we are only allowed to use f and f', this leads to the *Method of False Position* (see [334] or Problem 6.3).

Moreover, if we may compute the second derivative as well, we can use (6.11) to obtain a quadratic approximation of f and use the latter to find the minimum of f.

Newton's Method

This is commonly referred to as *Newton's method* (see Section 16.4.1 for a practical application of the latter to classification problems). We expand $f(x)$ around x_0;

$$f(x) \approx f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2} f''(x_0). \tag{6.12}$$

Minimization of the expansion (6.12) yields

$$x = x_0 - \frac{f'(x_0)}{f''(x_0)}. \tag{6.13}$$

Hence, we hope that if the approximation (6.12) is good, we will obtain an algorithm with fast convergence (Algorithm 6.2). Let us analyze the situation in more detail. For convenience, we state the result in terms of $g := f'$, since finding a zero of g is equivalent to finding a minimum of f.

Theorem 6.14 (Convergence of Newton Method) *Let $g : \mathbb{R} \to \mathbb{R}$ be a twice continuously differentiable function, and denote by $x^* \in \mathbb{R}$ a point with $g'(x^*) \neq 0$ and $g(x^*) = 0$. Then, provided x_0 is sufficiently close to x^*, the sequence generated by (6.13) will converge to x^* at least quadratically.*

Quadratic Convergence

Proof For convenience, denote by x_n the value of x at the nth iteration. As before, we apply Theorem 6.13. We now expand $g(x^*)$ around x_n. For some $\xi \in [0, x^* - x_n]$, we have

$$g(x_n) = g(x_n) - g(x^*) = g(x_n) - \left[g(x_n) + g'(x_n)(x^* - x_n) + \frac{\xi^2}{2} g''(x_n) \right], \tag{6.14}$$

and therefore by substituting (6.14) into (6.13),

$$x_{n+1} - x^* = x_n - x^* - \frac{g(x_n)}{g'(x_n)} = \xi^2 \frac{g''(x_n)}{2g'(x_n)}. \tag{6.15}$$

Since by construction $|\xi| \leq |x_n - x^*|$, we obtain a quadratically convergent algorithm in $|x_n - x^*|$, provided that $\left| (x_n - x^*) \frac{g''(x_n)}{2g'(x_n)} \right| < 1$. ∎

Region of Convergence

In other words, if the Newton method converges, it converges more rapidly than interval cutting or similar methods. We cannot guarantee beforehand that we are really in the region of convergence of the algorithm. In practice, if we apply the Newton method and find that it converges, we know that the solution has converged to the minimizer of f. For more information on optimization algorithms for unconstrained problems see [173, 530, 334, 15, 159, 45].

Line Search

In some cases we will not know an upper bound on the size of the interval to be analyzed for the presence of minima. In this situation we may, for instance, start with an initial guess of an interval, and if no minimum can be found strictly *inside* the interval, enlarge it, say by doubling its size. See [334] for more information on this matter. Let us now proceed to a technique which is quite popular (albeit not always preferable) in machine learning.

6.2.2 Functions of Several Variables: Gradient Descent

Gradient descent is one of the simplest optimization techniques to implement for minimizing functions of the form $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ may be $\mathbb{R}^N$, or indeed any set on which a gradient may be defined and evaluated. In order to avoid further complications we assume that the gradient $f'(x)$ exists and that we are able to compute it.

Direction of Steepest Descent

The basic idea is as follows: given a location x_n at iteration n, compute the gradient $g_n := f'(x_n)$, and update

$$x_{n+1} = x_n - \gamma g_n \tag{6.16}$$

such that the decrease in f is maximal over all $\gamma > 0$. For the final step, one of the algorithms from Section 6.2.1 can be used. It is straightforward to show that $f(x_n)$ is a monotonically decreasing series, since at each step the line search updates x_{n+1} in such a way that $f(x_{n+1}) < f(x_n)$. Such a value of γ must exist, since (again by Theorem 6.13) we may expand $f(x_n + \gamma g_n)$ in terms of γ around x_n, to obtain[1]

$$f(x_n - \gamma g_n) = f(x_n) - \gamma \|g_n\|^2 + O(\gamma^2). \tag{6.17}$$

As usual $\|\cdot\|$ is the Euclidean norm. For small γ the linear contribution in the Taylor expansion will be dominant, hence for some $\gamma > 0$ we have $f(x_n - \gamma g_n) < f(x_n)$. It can be shown [334] that after a (possibly infinite) number of steps, gradient descent (see Algorithm 6.3) will converge.

Problems of Convergence

In spite of this, the performance of gradient descent is far from optimal. Depending on the shape of the landscape of values of f, gradient descent may take a long time to converge. Figure 6.6 shows two examples of possible convergence behavior of the gradient descent algorithm.

1. To see that Theorem 6.13 applies in (6.17), note that $f(x_n + \gamma g_n)$ is a mapping $\mathbb{R} \to \mathbb{R}$ when viewed as a function of γ.

Algorithm 6.3 Gradient Descent

Require: x_0, Precision ϵ
  $n = 0$
  repeat
    Compute $g = f'(x_n)$
    Perform line search on $f(x_n - \gamma g)$ for optimal γ.
    $x_{n+1} = x_n - \gamma g$
    $n = n + 1$
  until $\|f'(x_n)\| \leq \epsilon$
Output: x_n

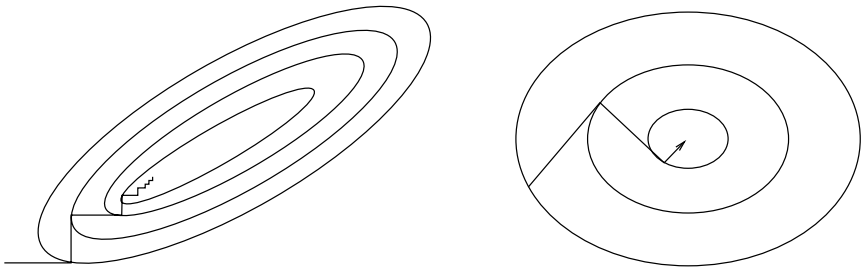


Figure 6.6 Left: Gradient descent takes a long time to converge, since the landscape of values of f forms a long and narrow valley, causing the algorithm to zig-zag along the walls of the valley. Right: due to the homogeneous structure of the minimum, the algorithm converges after very few iterations. Note that in both cases, the next direction of descent is *orthogonal* to the previous one, since line search provides the optimal step length.

6.2.3 Convergence Properties of Gradient Descent

Let us analyze the convergence properties of Algorithm 6.3 in more detail. To keep matters simple, we assume that f is a quadratic function, i.e.

$$f(x) = \frac{1}{2}(x - x^*)^\top K(x - x^*) + c_0, \tag{6.18}$$

where K is a positive definite symmetric matrix (cf. Definition 2.4) and c_0 is constant.[2] This is clearly a convex function with minimum at x^*, and $f(x^*) = c_0$. The gradient of f is given by

$$g := f'(x) = K(x - x^*). \tag{6.19}$$

To find the update of the steepest descent we have to minimize

$$f(x - \gamma g) = \frac{1}{2}(x - \gamma g - x^*)K(x - \gamma g - x^*) = \frac{1}{2}\gamma^2 g^\top K g - \gamma g^\top g. \tag{6.20}$$

2. Note that we may rewrite (up to a constant) any convex quadratic function $f(x) = x^\top K x + c^\top x + d$ in the form (6.18), simply by expanding f around its minimum value x^*.

By minimizing (6.20) for γ, the update of steepest descent is given explicitly by

$$x_{n+1} = x_n - \frac{g^\top g}{g^\top K g} g. \tag{6.21}$$

Improvement per Step

Substituting (6.21) into (6.18) and subtracting the terms $f(x_n)$ and $f(x_{n+1})$ yields the following improvement after an update step

$$\begin{aligned} f(x_n) - f(x_{n+1}) &= (x_n - x^*)^\top K \frac{g^\top g}{g^\top K g} g - \frac{1}{2}\left(\frac{g^\top g}{g^\top K g}\right)^2 g^\top K g \\ &= \frac{1}{2}\frac{(g^\top g)^2}{g^\top K g} = f(x_n)\left[\frac{(g^\top g)^2}{(g^\top K g)(g^\top K^{-1} g)}\right]. \end{aligned} \tag{6.22}$$

Thus the relative improvement per iteration depends on the value of $t(g) := \frac{(g^\top g)^2}{(g^\top K g)(g^\top K^{-1} g)}$. In order to give performance guarantees we have to find a lower bound for $t(g)$. To this end we introduce the *condition* of a matrix.

Definition 6.15 (Condition of a Matrix) *Denote by K a matrix and by $\lambda_{\max}$ and $\lambda_{\min}$ its largest and smallest singular values (or eigenvalues if they exist) respectively. The condition of a matrix is defined as*

$$\text{cond } K := \frac{\lambda_{\max}}{\lambda_{\min}}. \tag{6.23}$$

Clearly, as cond K decreases, different directions are treated in a more homogeneous manner by $x^\top K x$. In particular, note that smaller cond K correspond to less elliptic contours in Figure 6.6. Kantorovich proved the following inequality which allows us to connect the condition number with the convergence behavior of gradient descent algorithms.

Theorem 6.16 (Kantorovich Inequality [278]) *Denote by $K \in \mathbb{R}^{m \times m}$ (typically the kernel matrix) a strictly positive definite symmetric matrix with largest and smallest eigenvalues $\lambda_{\max}$ and $\lambda_{\min}$. Then the following inequality holds for any $g \in \mathbb{R}^m$:*

Lower Bound for Improvement

$$\frac{(g^\top g)^2}{(g^\top K g)(g^\top K^{-1} g)} \geq \frac{4\lambda_{\min}\lambda_{\max}}{(\lambda_{\min} + \lambda_{\max})^2} \geq \frac{1}{\text{cond } K}. \tag{6.24}$$

We typically denote by g the gradient of f. The second inequality follows immediately from Definition 6.15; the proof of the first inequality is more technical, and is not essential to the understanding of the situation. See Problem 6.7 and [278, 334] for more detail.

A brief calculation gives us the correct order of magnitude. Note that for any x, the quadratic term $x^\top K x$ is bounded from above by $\lambda_{\max}\|x\|^2$, and likewise $x^\top K^{-1} x \leq \lambda_{\min}^{-1}\|x\|^2$. Hence we bound the relative improvement $t(g)$ (as defined below (6.22)) by $1/(\text{cond } K)$ which is almost as good as the second term in (6.24) (the latter can be up to a factor of 4 better for $\lambda_{\min} \ll \lambda_{\max}$).

This means that gradient descent methods perform poorly if some of the eigenvalues of K are very small in comparison with the largest eigenvalue, as is usually the case with matrices generated by positive definite kernels (and as sometimes

desired for learning theoretical reasons); see Chapter 4 for details. This is one of the reasons why many gradient descent algorithms for training Support Vector Machines, such as the Kernel AdaTron [183, 12] or AdaLine [185], exhibit poor convergence. Section 10.6.1 deals with these issues, and sets up the gradient descent directions both in the Reproducing Kernel Hilbert Space $\mathcal{H}$ and in coefficient space $\mathbb{R}^m$.

6.2.4 Functions of Several Variables: Conjugate Gradient Descent

Let us now look at methods that are better suited to minimizing convex functions. Again, we start with quadratic forms. The key problem with gradient descent is that the quotient between the smallest and the largest eigenvalue can be very large, which leads to slow convergence. Hence, one possible technique is to *rescale* $\mathcal{X}$ by some matrix M such that the condition of $K \in \mathbb{R}^{m\times m}$ in this rescaled space, which is to say the condition of $M^\top KM$, is much closer to 1 (in numerical analysis this is often referred to as *preconditioning* [247, 423, 530]). In addition, we would like to focus first on the largest eigenvectors of K.

A key tool is the concept of *conjugate directions*. The basic idea is that rather than using the metric of the normal dot product $x^\top x' = x^\top \mathbf{1} x'$ ($\mathbf{1}$ is the unit matrix) we use the metric imposed by K, i.e. $x^\top Kx'$, to guide our algorithm, and we introduce an equivalent notion of orthogonality with respect to the new metric.

Definition 6.17 (Conjugate Directions) *Given a symmetric matrix $K \in \mathbb{R}^{m\times m}$, any two vectors $v, v' \in \mathbb{R}^m$ are called K-orthogonal if $v^\top Kv' = 0$.*

Likewise, we can introduce notions of a basis and of linear independence with respect to K. The following theorem establishes the necessary identities.

Theorem 6.18 (Orthogonal Decompositions in K) *Denote by $K \in \mathbb{R}^{m\times m}$ a strictly positive definite symmetric matrix and by $v_1, \ldots, v_m$ a set of mutually K-orthogonal and nonzero vectors. Then the following properties hold:*

(i) The vectors $v_1, \ldots, v_m$ form a basis.

(ii) Any $x \in \mathbb{R}^m$ can be expanded in terms of v_i by

$$x = \sum_{i=1}^{m} v_i \frac{v_i^\top Kx}{v_i^\top Kv_i}. \tag{6.25}$$

In particular, for any $y = Kx$, we can find x by

$$x = \sum_{i=1}^{m} v_i \frac{v_i^\top y}{v_i^\top Kv_i}. \tag{6.26}$$

Linear Independence

Proof (i) Since we have m vectors in $\mathbb{R}^m$, all we have to show is that the vectors v_i are linearly independent. Assume that there exist some $\alpha_i \in \mathbb{R}$ such that $\sum_{i=1}^m \alpha_i v_i =$

0. Then due to K-orthogonality, we have

$$0 = v_j^\top K \left[\sum_{i=1}^m \alpha_i v_i \right] = \sum_{i=1}^m \alpha_i v_j^\top K v_i = \alpha_j v_j^\top K v_j \text{ for all } j. \tag{6.27}$$

Hence $\alpha_j = 0$ for all j. This means that all v_j are linearly independent.

(ii) The vectors $\{v_1, \ldots, v_m\}$ form a basis. Therefore we may expand any $x \in \mathbb{R}^m$ as a linear combination of v_j, i.e. $x = \sum_{i=1}^m \alpha_i v_i$. Consequently we can expand $v_j^\top K x$ in terms of $v_j^\top K v_i$, and we obtain

$$v_j^\top K x = v_j^\top K \left[\sum_{i=1}^m \alpha_i v_i \right] = \alpha_j v_j^\top K v_j. \tag{6.28}$$

Basis Expansion

Solving for α_j proves the claim.

(iii) Let $y = Kx$. Since the vectors v_i form a basis, we can expand x in terms of α_i. Substituting this definition into (6.28) proves (6.26). ∎

The practical consequence of this theorem is that, provided we know a set of K-orthogonal vectors v_i, we can solve the linear equation $y = Kx$ via (6.26). Furthermore, we can also use it to minimize quadratic functions of the form $f(x) = \frac{1}{2} x^\top K x - c^\top x$. The following theorem tells us how.

Theorem 6.19 (Deflation Method) *Denote by $v_1, \ldots, v_m$ a set of mutually K-orthogonal vectors for a strictly positive definite symmetric matrix $K \in \mathbb{R}^{m \times m}$. Then for any $x_0 \in \mathbb{R}^m$ the following method finds x_i that minimize $f(x) = \frac{1}{2} x^\top K x - c^\top x$ in the linear manifold $\mathcal{X}_i := x_0 + \operatorname{span}\{v_1, \ldots, v_i\}$.*

Optimality in Linear Space

$$x_i := x_{i-1} - v_i \frac{g_{i-1}^\top v_i}{v_i^\top K v_i} \text{ where } g_{i-1} = f'(x_{i-1}) \text{ for all } i > 0. \tag{6.29}$$

Proof We use induction. For $i = 0$ the statement is trivial, since the linear manifold consists of only one point.

Assume that the statement holds for i. Since f is convex, we only need prove that the gradient of $f(x_i)$ is orthogonal to $\operatorname{span}\{v_1, \ldots, v_i\}$. In that case no further improvement can be gained on the linear manifold $\mathcal{X}_i$. It suffices to show that for all $j \leq i + 1$,

$$0 = v_j^\top g_i. \tag{6.30}$$

Gradient Descent in Rescaled Space

Additionally, we may expand x_{i+1} to obtain

$$v_j^\top g_i = v_j^\top \left[K x_{i-1} - c - K v_i \frac{g_{i-1}^\top v_i}{v_i^\top K v_i} \right] = v_j^\top g_{i-1} - (g_{i-1}^\top v_i) \frac{v_j^\top K v_i}{v_i^\top K v_i}. \tag{6.31}$$

For $j = i$ both terms cancel out. For $j < i$ both terms vanish due to the induction assumption. Since the vectors v_j form a basis $\mathcal{X}_m = \mathbb{R}^m$, x_m is a minimizer of f. ∎

In a nutshell, Theorem 6.19 already contains the Conjugate Gradient descent al-

Algorithm 6.4 Conjugate Gradient Descent

Require: x_0
  Set $i = 0$
  $g_0 = f'(x_0)$
  $v_0 = g_0$
  repeat
    $x_{i+1} = x_i + \alpha_i v_i$ where $\alpha_i = -\frac{g_i^\top v_i}{v_i^\top K v_i}$
    $g_{i+1} = f'(x_{i+1})$
    $v_{i+1} = -g_{i+1} + \beta_i v_i$ where $\beta_i = \frac{g_{i+1}^\top K v_i}{v_i^\top K v_i}$.
    $i = i + 1$
  until $g_i = 0$
Output: x_i

gorithm: in each step we perform gradient descent with respect to one of the K-orthogonal vectors v_i, which means that after n steps we will reach the minimum. We still lack a method to obtain such a K-orthogonal basis of vectors v_i. It turns out that we can get the latter directly from the gradients g_i. Algorithm 6.4 describes the procedure.

All we have to do is prove that Algorithm 6.4 actually does what it is required to do, namely generate a K-orthogonal set of vectors v_i, and perform deflation in the latter. To achieve this, the v_i are obtained by an orthogonalization procedure akin to Gram-Schmidt orthogonalization.

Theorem 6.20 (Conjugate Gradient) *Assume we are given a quadratic convex function $f(x) = \frac{1}{2}x^\top Kx - c^\top x$, to which we apply conjugate gradient descent for minimization purposes. Then algorithm 6.4 is a deflation method, and unless $g_i = 0$, we have for every $0 \leq i \leq m$,*

(i) $\operatorname{span}\{g_0, \ldots, g_i\} = \operatorname{span}\{v_0, \ldots, v_i\} = \operatorname{span}\{g_0, Kg_0, \ldots, K^i g_0\}$.

(ii) The vectors v_i are K-orthogonal.

(iii) The equations in Algorithm 6.4 for α_i and β_i can be replaced by $\alpha_i = \frac{g_i^\top g_i}{v_i^\top K v_i}$ and $\beta_i = \frac{g_{i+1}^\top g_{i+1}}{g_i^\top g_i}$.

(iv) After i steps, x_i is the solution in the manifold $x_0 + \operatorname{span}\{g_0, Kg_0, \ldots, K^{i-1}g_0\}$.

Proof **(i) and (ii)** We use induction. For $i = 0$ the statements trivially hold since $v_0 = g_0$. For i note that by construction (see Algorithm 6.4) $g_{i+1} = Kx_{i+1} - c = g_i + \alpha_i K v_i$, hence $\operatorname{span}\{g_0, \ldots, g_{i+1}\} = \operatorname{span}\{g_0, Kg_0, \ldots, K^{i+1}g_0\}$. Since $v_{i+1} = -g_{i+1} + \beta_i v_i$ the same statement holds for $\operatorname{span}\{v_0, \ldots, v_{i+1}\}$. Moreover, the vectors g_i are linearly independent or 0 due to Theorem 6.19.
Finally $v_j^\top K v_{i+1} = -v_j^\top K g_{i+1} + \beta_i v_j^\top K v_i = 0$, since for $j = i$ both terms cancel out, and for $j < i$ both terms individually vanish (due to Theorem 6.19 and (i)).

(iii) We have $-g_i^\top v_i = g_i^\top g_i - \beta_{i-1} g_i^\top v_{i-1} = g_i^\top g_i$, since the second term vanishes due to Theorem 6.19. This proves the result for α_i.

Table 6.1 Non-quadratic modifications of conjugate gradient descent.

Generic Method	Compute Hessian $K_i := f''(x_i)$ and update α_i, β_i with $\alpha_i = -\frac{g_i^\top v_i}{v_i^\top K_i v_i}$ $\beta_i = \frac{g_{i+1}^\top K_i v_i}{v_i^\top K_i v_i}$ This requires calculation of the Hessian at each iteration.
Fletcher-Reeves [173]	Find α_i via a line search and use Theorem 6.20 (iii) for β_i $\alpha_i = \operatorname{argmin}_\alpha f(x_i + \alpha v_i)$ $\beta_i = \frac{g_{i+1}^\top g_{i+1}}{g_i^\top g_i}$
Polak-Ribiere [414]	Find α_i via a line search $\alpha_i = \operatorname{argmin}_\alpha f(x_i + \alpha v_i)$ $\beta_i = \frac{(g_{i+1}-g_i)^\top g_{i+1}}{g_i^\top g_i}$ Experimentally, Polak-Ribiere tends to be better than Fletcher-Reeves.

For β_i note that $g_{i+1}^\top K v_i = \alpha_i^{-1} g_{i+1}^\top (g_{i+1} - g_i) = \alpha_i^{-1} g_{i+1}^\top g_{i+1}$. Substitution of the value of α_i proves the claim.

(iv) Again, we use induction. At step $i = 1$ we compute the solution within the space spanned by g_0. ∎

We conclude this section with some remarks on the optimality of conjugate gradient descent algorithms, and how they can be extended to arbitrary convex functions.

Space of Largest Eigenvalues

Due to Theorems 6.19 and 6.20, we can see that after i iterations, the conjugate gradient descent algorithm finds a solution on the linear manifold $x_0 + \operatorname{span}\{g_0, Kg_0, \ldots, K^{i-1}g_0\}$. This means that the solutions will be mostly aligned with the largest eigenvalues of K, since after multiple application of K to any arbitrary vector g_0, the largest eigenvectors dominate. Nonetheless, the algorithm here is significantly cheaper than computing the eigenvalues of K, and subsequently minimizing f in the subspace corresponding to the largest eigenvalues. For more detail see [334]

Nonlinear Extensions

In the case of general convex functions, the assumptions of Theorem 6.20 are no longer satisfied. In spite of this, conjugate gradient descent has proven to be effective even in these situations. Additionally, we have to account for some modifications. Basically, the update rules for g_i and v_i remain unchanged but the parameters α_i and β_i are computed differently. Table 6.1 gives an overview of different methods. See [173, 334, 530, 414] for details.

6.2.5 Predictor Corrector Methods

As we go to higher order Taylor expansions of the function f to be minimized (or set to zero), the corresponding numerical methods become increasingly com-

Increasing the Order

plicated to implement, and require an ever increasing number of parameters to be estimated or computed. For instance, a quadratic expansion of a multivariate function $f : \mathbb{R}^m \to \mathbb{R}$ requires $m \times m$ terms for the quadratic part (the Hessian), whereas the linear part (the gradient) can be obtained by computing m terms. Since the quadratic expansion is only an approximation for most non-quadratic functions, this is wasteful (for interior point programs, see Section 6.4). We might instead be able to achieve roughly the same goal without computing the quadratic term explicitly, or more generally, obtain the performance of higher order methods without actually implementing them.

This can in fact be achieved using predictor-corrector methods. These work by computing a tentative update $x_i \to x_{i+1}^{\text{pred}}$ (predictor step), then using x_{i+1}^{pred} to account for higher order changes in the objective function, and finally obtaining a *corrected* value x_{i+1}^{corr} based on these changes. A simple example illustrates the method. Assume we want to find the solution to the equation

Predictor Corrector Methods for Quadratic Equations

$$f(x) = 0 \text{ where } f(x) = f_0 + ax + \frac{1}{2}bx^2. \tag{6.32}$$

We assume $a, b, f_0, x \in \mathbb{R}$. Exact solution of (6.32) requires taking a square root. Let us see whether we can find an approximate method that avoids this (in general b will be an $m \times m$ matrix, so this is a worthwhile goal). The predictor corrector approach works as follows: first solve

$$f_0 + ax = 0 \text{ and hence } x^{\text{pred}} = -\frac{f_0}{a}. \tag{6.33}$$

Second, substitute x^{pred} into the nonlinear parts of (6.32) to obtain

$$f_0 + ax^{\text{corr}} + \frac{1}{2}b\left(\frac{f_0}{a}\right)^2 = 0 \text{ and hence } x^{\text{corr}} = -\frac{f_0}{a}\left(1 + \frac{1}{2}\frac{bf_0}{a^2}\right). \tag{6.34}$$

No Quadratic Residuals

Comparing x^{pred} and x^{corr}, we see that $\frac{1}{2}\frac{bf_0}{a^2}$ is the correction term that takes the effect of the changes in x into account.

Since neither of the two values (x^{pred} or x^{corr}) will give us the exact solution to $f(x) = 0$ in just one step, it is worthwhile having a look at the errors of both approaches.

$$f(x^{\text{pred}}) = \frac{1}{2}\frac{bf_0^2}{a^2} \text{ and } f(x^{\text{corr}}) = 2\frac{f^2(x^{\text{pred}})}{f_0} + \frac{f^3(x^{\text{pred}})}{f_0^2}. \tag{6.35}$$

We can check that if $\frac{bf_0}{a^2} \leq 2 - 2\sqrt{2}$, the corrector estimate will be better than the predictor one. As our initial estimate f_0 decreases, this will be the case. Moreover, we can see that $f(x^{\text{corr}})$ only contains terms in x that are of higher order than quadratic. This means that even though we did not solve the quadratic form explicitly, we eliminated all corresponding terms.

The general scheme is described in Algorithm 6.5. It is based on the assumption that $f(x + \xi)$ can be split up into

$$f(x + \xi) = f(x) + f_{\text{simple}}(\xi, x) + T(\xi, x), \tag{6.36}$$

Algorithm 6.5 Predictor Corrector Method

Require: x_0, Precision ϵ
  Set $i = 0$
  repeat
    Expand f into $f(x_i) + f_{\text{simple}}(\xi, x_i) + T(\xi, x_i)$.
Predictor Solve $f(x_i) + f_{\text{simple}}(\xi^{\text{pred}}, x_i) = 0$ for ξ^{pred}.
Corrector Solve $f(x_i) + f_{\text{simple}}(\xi^{\text{corr}}, x_i) + T(\xi^{\text{pred}}, x_i) = 0$ for ξ^{corr}.
    $x_{i+1} = x_i + \xi^{\text{corr}}$.
    $i = i + 1$.
  until $|f(x_i)| \leq \epsilon$
Output: x_i

where $f_{\text{simple}}(\xi, x)$ contains the simple, possibly low order, part of f, and $T(\xi, x)$ the higher order terms, such that $f_{\text{simple}}(0, x) = T(0, x) = 0$. While in the previous example we introduced higher order terms into f that were not present before (f is only quadratic), usually such terms will already exist anyway. Hence the corrector step will just eliminate additional lower order terms without too much additional error in the approximation.
We will encounter such methods for instance in the context of interior point algorithms (Section 6.4), where we have to solve a set of quadratic equations.

6.3 Constrained Problems

After this digression on unconstrained optimization problems, let us return to constrained optimization, which makes up the main body of the problems we will have to deal with in learning (e.g., quadratic or general convex programs for Support Vector Machines). Typically, we have to deal with problems of type (6.6). For convenience we repeat the problem statement:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & c_i(x) \leq 0 \text{ for all } i \in [n]. \end{array} \tag{6.37}$$

Here f and c_i are convex functions and $n \in \mathbb{N}$. In some cases[3], we additionally have *equality* constraints $e_j(x) = 0$ for some $j \in [n']$. Then the optimization problem can be written as

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f(x), \\ \text{subject to} & c_i(x) \leq 0 \text{ for all } i \in [n], \\ & e_j(x) = 0 \text{ for all } j \in [n']. \end{array} \tag{6.38}$$

3. Note that it is common practice in Support Vector Machines to write c_i as positivity constraints by using concave functions. This can be fixed by a sign change, however.

Algorithm 6.5 Predictor Corrector Method

Require: x_0, Precision ϵ
  Set $i = 0$
  repeat
    Expand f into $f(x_i) + f_{\text{simple}}(\xi, x_i) + T(\xi, x_i)$.
Predictor Solve $f(x_i) + f_{\text{simple}}(\xi^{\text{pred}}, x_i) = 0$ for ξ^{pred}.
Corrector Solve $f(x_i) + f_{\text{simple}}(\xi^{\text{corr}}, x_i) + T(\xi^{\text{pred}}, x_i) = 0$ for ξ^{corr}.
    $x_{i+1} = x_i + \xi^{\text{corr}}$.
    $i = i + 1$.
  until $|f(x_i)| \leq \epsilon$
Output: x_i

where $f_{\text{simple}}(\xi, x)$ contains the simple, possibly low order, part of f, and $T(\xi, x)$ the higher order terms, such that $f_{\text{simple}}(0, x) = T(0, x) = 0$. While in the previous example we introduced higher order terms into f that were not present before (f is only quadratic), usually such terms will already exist anyway. Hence the corrector step will just eliminate additional lower order terms without too much additional error in the approximation.
We will encounter such methods for instance in the context of interior point algorithms (Section 6.4), where we have to solve a set of quadratic equations.

6.3 Constrained Problems

After this digression on unconstrained optimization problems, let us return to constrained optimization, which makes up the main body of the problems we will have to deal with in learning (e.g., quadratic or general convex programs for Support Vector Machines). Typically, we have to deal with problems of type (6.6). For convenience we repeat the problem statement:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & c_i(x) \leq 0 \text{ for all } i \in [n]. \end{array} \tag{6.37}$$

Here f and c_i are convex functions and $n \in \mathbb{N}$. In some cases[3], we additionally have *equality* constraints $e_j(x) = 0$ for some $j \in [n']$. Then the optimization problem can be written as

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f(x), \\ \text{subject to} & c_i(x) \leq 0 \text{ for all } i \in [n], \\ & e_j(x) = 0 \text{ for all } j \in [n']. \end{array} \tag{6.38}$$

3. Note that it is common practice in Support Vector Machines to write c_i as positivity constraints by using concave functions. This can be fixed by a sign change, however.

Before we start minimizing f, we have to discuss what optimality means in this case. Clearly $f'(x) = 0$ is too restrictive a condition. For instance, f' could point into a direction which is forbidden by the constraints c_i and e_i. Then we could have optimality, even though $f' \neq 0$. Let us analyze the situation in more detail.

6.3.1 Optimality Conditions

We start with optimality conditions for optimization problems which are independent of their differentiability. While it is fairly straightforward to state *sufficient* optimality conditions for arbitrary functions f and c_i, we will need convexity and "reasonably nice" constraints (see Lemma 6.23) to state *necessary* conditions. This is not a major concern, since for practical applications, the constraint qualification criteria are almost always satisfied, and the functions themselves are usually convex and differentiable. Much of the reasoning in this section follows [345], which should also be consulted for further references and detail.

Some of the most important sufficient criteria are the Kuhn-Tucker[4] saddle point conditions [312]. As indicated previously, they are independent of assumptions on convexity or differentiability of the constraints c_i or objective function f.

Theorem 6.21 (Kuhn-Tucker Saddle Point Condition [312, 345]) *Assume an optimization problem of the form (6.37), where $f : \mathbb{R}^m \to \mathbb{R}$ and $c_i : \mathbb{R}^m \to \mathbb{R}$ for $i \in [n]$ are arbitrary functions, and a Lagrangian*

Lagrangian

$$L(x, \alpha) := f(x) + \sum_{i=1}^{n} \alpha_i c_i(x) \text{ where } \alpha_i \geq 0. \tag{6.39}$$

If a pair of variables $(\bar{x}, \bar{\alpha})$ with $\bar{x} \in \mathbb{R}^n$ and $\bar{\alpha}_i \geq 0$ for all $i \in [n]$ exists, such that for all $x \in \mathbb{R}^m$ and $\alpha \in [0, \infty)^n$,

$$L(\bar{x}, \alpha) \leq L(\bar{x}, \bar{\alpha}) \leq L(x, \bar{\alpha}) \text{ (Saddle Point)} \tag{6.40}$$

then $\bar{x}$ is a solution to (6.37).

The parameters α_i are called Lagrange multipliers. As described in the later chapters, they will become the coefficients in the kernel expansion in SVM.

Proof The proof follows [345]. Denote by $(\bar{x}, \bar{\alpha})$ a pair of variables satisfying (6.40). From the first inequality it follows that

$$\sum_{i=1}^{n} (\alpha_i - \bar{\alpha}_i) c_i(\bar{x}) \leq 0. \tag{6.41}$$

Since we are free to choose $\alpha_i \geq 0$, we can see (by setting all but one of the terms α_i to $\bar{\alpha}_i$ and the remaining one to $\alpha_i = \bar{\alpha}_i + 1$) that $c_i(x) \leq 0$ for all $i \in [n]$. This shows that $\bar{x}$ satisfies the constraints, i.e. it is feasible.

4. An earlier version is due to Karush [283]. This is why often one uses the abbreviation KKT (Karush-Kuhn-Tucker) rather than KT to denote the optimality conditions.

Additionally, by setting one of the α_i to 0, we see that $\bar{\alpha}_i c_i(\bar{x}) \geq 0$. The only way to satisfy this is by having

$$\bar{\alpha}_i c_i(\bar{x}) = 0 \text{ for all } i \in [n]. \tag{6.42}$$

Eq. (6.42) is often referred to as the KKT condition [283, 312]. Finally, combining (6.42) and $c_i(x) \leq 0$ with the second inequality in (6.40) yields $f(\bar{x}) \leq f(x)$ for all feasible x. This proves that $\bar{x}$ is optimal. ■

We can immediately extend Theorem 6.21 to accommodate equality constraints by splitting them into the conditions $e_i(x) \leq 0$ and $e_i(x) \geq 0$. We obtain:

Theorem 6.22 (Equality Constraints) *Assume an optimization problem of the form (6.38), where $f, c_i, e_j : \mathbb{R}^m \to \mathbb{R}$ for $i \in [n]$ and $j \in [n']$ are arbitrary functions, and a Lagrangian*

$$L(x, \alpha) := f(x) + \sum_{i=1}^{n} \alpha_i c_i(x) + \sum_{j=1}^{n'} \beta_j e_j(x) \text{ where } \alpha_i \geq 0 \text{ and } \beta_j \in \mathbb{R}. \tag{6.43}$$

If a set of variables $(\bar{x}, \bar{\alpha}, \bar{\beta})$ with $\bar{x} \in \mathbb{R}^m$, $\bar{\alpha} \in [0, \infty)$, and $\bar{\beta} \in \mathbb{R}^{n'}$ exists such that for all $x \in \mathbb{R}^m$, $\alpha \in [0, \infty)^n$, and $\beta \in \mathbb{R}^{n'}$,

$$L(\bar{x}, \alpha, \beta) \leq L(\bar{x}, \bar{\alpha}, \bar{\beta}) \leq L(x, \bar{\alpha}, \bar{\beta}), \tag{6.44}$$

then $\bar{x}$ is a solution to (6.38).

Now we determine when the conditions of Theorem 6.21 are necessary. We will see that convexity and sufficiently "nice" constraints are needed for (6.40) to become a necessary condition. The following lemma (see [345]) describes three *constraint qualifications*, which will turn out to be exactly what we need.

Feasible Region

Lemma 6.23 (Constraint Qualifications) *Denote by $\mathcal{X} \subset \mathbb{R}^m$ a convex set, and by $c_1, \ldots, c_n : \mathcal{X} \to \mathbb{R}$ n convex functions defining a feasible region by*

$$X := \{x | x \in \mathcal{X} \text{ and } c_i(x) \leq 0 \text{ for all } i \in [n]\}. \tag{6.45}$$

Equivalence Between Constraint Qualifications

Then the following additional conditions on c_i are connected by $(i) \iff (ii)$ and $(iii) \implies (i)$.

(i) There exists an $x \in \mathcal{X}$ such that for all $i \in [n]$ $c_i(x) < 0$ (Slater's condition [500]).

(ii) For all nonzero $\alpha \in [0, \infty)^n$ there exists an $x \in \mathcal{X}$ such that $\sum_{i=1}^{n} \alpha_i c_i(x) \leq 0$ (Karlin's condition [281]).

(iii) The feasible region X contains at least two distinct elements, and there exists an $x \in X$ such that all c_i are strictly convex at x wrt. X (Strict constraint qualification).

The connection $(i) \iff (ii)$ is also known as the Generalized Gordan Theorem [164]. The proof can be skipped if necessary. We need an auxiliary lemma which we state without proof (see [345, 435] for details).

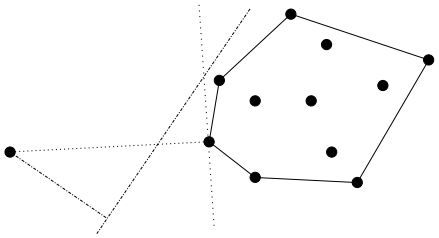


Figure 6.7 Two hyperplanes (and their normal vectors) separating the convex hull of a finite set of points from the origin.

Lemma 6.24 (Separating Hyperplane Theorem) *Denote by $X \in \mathbb{R}^m$ a convex set not containing the origin 0. Then there exists a hyperplane with normal vector $\alpha \in \mathbb{R}^m$ such that $\alpha^\top x > 0$ for all $x \in X$.*

See also Figure 6.7.

Proof of Lemma 6.23. We prove $\{(i) \iff (ii)\}$ by showing $\{(i) \implies (ii)\}$ and $\{$ not $(i) \implies$ not $(ii)\}$.

$(i) \implies (ii)$ For a point $x \in X$ with $c_i(x) < 0$, for all $i \in [n]$ we have that $\alpha_i c_i(x) \geq 0$ implies $\alpha_i = 0$.

$\overline{(i)} \implies \overline{(ii)}$ Assume that there is no x with $c_i(x) < 0$ for all $i \in [n]$. Hence the set

$$\Gamma := \{\gamma | \gamma \in \mathbb{R}^n \text{ and there exists some } x \in X \text{ with } \gamma_i > c_i(x) \text{ for all } i \in [n]\} \tag{6.46}$$

is convex and does not contain the origin. The latter follows directly from the assumption. For the former take $\gamma, \gamma' \in \Gamma$ and $\lambda \in (0,1)$ to obtain

$$\lambda\gamma_i + (1-\lambda)\gamma_i' > \lambda c_i(x) + (1-\lambda)c_i(x') \geq c_i(\lambda x + (1-\lambda)x'). \tag{6.47}$$

Now by Lemma 6.24, there exists some $\alpha \in \mathbb{R}^n$ such that $\alpha^\top \gamma \geq 0$ and $\|\alpha\|^2 = 1$ for all $\gamma \in \Gamma$. Since each of the γ_i for $\gamma \in \Gamma$ can be arbitrarily large (with respect to the other coordinates), we conclude $\alpha_i \geq 0$ for all $i \in [n]$.
Denote by $\delta := \inf_{x \in X} \sum_{i=1}^n \alpha_i c_i(x)$ and by $\delta' := \inf_{\gamma \in \Gamma} \alpha^\top \gamma$. One can see that by construction $\delta = \delta'$. By Lemma 6.24 α was chosen such that $\delta' \geq 0$, and hence $\delta \geq 0$. This contradicts (ii), however, since it implies the existence of a suitable α with $\alpha_i c_i(x) \geq 0$ for all x.

$(iii) \implies (i)$ Since X is convex we get for all c_i and for any $\lambda \in (0,1)$:

$$\lambda x + (1-\lambda)x' \in X \text{ and } 0 \geq \lambda c_i(x) + (1-\lambda)c_i(x') > c_i(\lambda x + (1-\lambda)x'). \tag{6.48}$$

This shows that $\lambda x + (1-\lambda)x'$ satisfies (i) and we are done.

■

We proved Lemma 6.23 as it provides us with a set of constraint qualifications (conditions on the constraints) that allow us to determine cases where the KKT saddle point conditions are both *necessary* and sufficient. This is important, since we will use the KKT conditions to transform optimization problems into their duals, and solve the latter numerically. For this approach to be valid, however, we must ensure that we do not change the solvability of the optimization problem.

Theorem 6.25 (Necessary KKT Conditions [312, 553, 281]) *Under the assumptions and definitions of Theorem 6.21 with the additional assumption that f and c_i are convex on the convex set $X \subseteq \mathbb{R}^m$ (containing the set of feasible solutions as a subset) and that c_i satisfy one of the constraint qualifications of Lemma 6.23, the saddle point criterion (6.40) is necessary for optimality.*

Proof Denote by $\bar{x}$ the solution to (6.37), and by X' the set

$$X' := X \cap \{x | x \in \mathcal{X} \text{ with } f(x) - f(\bar{x}) \leq 0 \text{ and } c_i(x) \leq 0 \text{ for all } i \in [n]\}. \tag{6.49}$$

By construction $\bar{x} \in X'$. Furthermore, there exists no $x' \in X'$ such that all inequality constraints including $f(x) - f(\bar{x})$ are satisfied as *strict* inequalities (otherwise $\bar{x}$ would not be optimal). In other words, X' violates Slater's conditions (i) of Lemma 6.23 (where both $\big(f(x) - f(\bar{x})\big)$ and $c(x)$ *together* play the role of $c_i(x)$), and thus also Karlin's conditions (ii). This means that there exists a nonzero vector $(\bar{\alpha}_0, \bar{\alpha}) \in \mathbb{R}^{n+1}$ with nonnegative entries such that

$$\bar{\alpha}_0(f(x) - f(\bar{x})) + \sum_{i=1}^{n} \bar{\alpha}_i c_i(x) \geq 0 \text{ for all } x \in \mathcal{X}. \tag{6.50}$$

In particular, for $x = \bar{x}$ we get $\sum_{i=1}^{n} \bar{\alpha}_i c_i(\bar{x}) \geq 0$. In addition, since $\bar{x}$ is a solution to (6.37), we have $c_i(\bar{x}) \leq 0$. Hence $\sum_{i=1}^{n} \bar{\alpha}_i c_i(\bar{x}) = 0$. This allows us to rewrite (6.50) as

$$\bar{\alpha}_0 f(x) + \sum_{i=1}^{n} \bar{\alpha}_i c_i(x) \geq \bar{\alpha}_0 f(\bar{x}) + \sum_{i=1}^{n} \bar{\alpha}_i c_i(\bar{x}). \tag{6.51}$$

This looks almost like the first inequality of (6.40), except for the $\bar{\alpha}_0$ term (which we will return to later). But let us consider the second inequality first.

Again, since $c_i(\bar{x}) \leq 0$ we have $\sum_{i=1}^{n} \alpha_i c_i(\bar{x}) \leq 0$ for all $\alpha_i \geq 0$. Adding $\bar{\alpha}_0 f(\bar{x})$ on both sides of the inequality and $\sum_{i=1}^{n} \bar{\alpha}_i c_i(\bar{x})$ on the rhs yields

$$\bar{\alpha}_0 f(\bar{x}) + \sum_{i=1}^{n} \bar{\alpha}_i c_i(\bar{x}) \geq \bar{\alpha}_0 f(\bar{x}) + \sum_{i=1}^{n} \alpha_i c_i(\bar{x}). \tag{6.52}$$

This is almost all we need for the first inequality of (6.40) .[5] If $\bar{\alpha}_0 > 0$ we can divide (6.51) and (6.52) by $\bar{\alpha}_0$ and we are done.

When $\bar{\alpha}_0 = 0$, then this implies the existence of $\bar{\alpha} \in \mathbb{R}^n$ with nonnegative entries satisfying $\sum_{i=1}^{n} \bar{\alpha}_i c_i(x) \geq 0$ for all $x \in X$. This contradicts Karlin's constraint qualification condition (ii), which allows us to rule out this case. ∎

6.3.2 Duality and KKT-Gap

Now that we have formulated necessary and sufficient optimality conditions (Theorem 6.21 and 6.25) under quite general circumstances, let us put them to practical

5. The two inequalities (6.51) and (6.52) are also known as the Fritz-John saddle point necessary optimality conditions [269], which play a similar role as the saddle point conditions for the Lagrangian (6.39) of Theorem 6.21.

use for convex differentiable optimization problems. We first derive a more practically useful form of Theorem 6.21. Our reasoning is as follows: eq. (6.40) implies that $L(\bar{x}, \bar{\alpha})$ is a *saddle point* in terms of $(\bar{x}, \bar{\alpha})$. Hence, all we have to do is write the saddle point conditions in the form of derivatives.

Theorem 6.26 (KKT for Differentiable Convex Problems [312]) *A solution to the optimization problem (6.37) with convex, differentiable f, c_i is given by $\bar{x}$, if there exists some $\bar{\alpha} \in \mathbb{R}^n$ with $\alpha_i \geq 0$ for all $i \in [n]$ such that the following conditions are satisfied:*

Primal and Dual Feasibility

$$\partial_x L(\bar{x}, \bar{\alpha}) = \partial_x f(\bar{x}) + \sum_{i=1}^{n} \bar{\alpha}_i \partial_x c_i(\bar{x}) = 0 \text{ (Saddle Point in } \bar{x}\text{)}, \tag{6.53}$$

$$\partial_{\alpha_i} L(\bar{x}, \bar{\alpha}) = c_i(\bar{x}) \leq 0 \text{ (Saddle Point in } \bar{\alpha}\text{)}, \tag{6.54}$$

$$\sum_{i=1}^{n} \bar{\alpha}_i c_i(\bar{x}) = 0 \text{ (Vanishing KKT-Gap)}. \tag{6.55}$$

Proof The easiest way to prove Theorem 6.26 is to show that for any $x \in X$, we have $f(x) - f(\bar{x}) \geq 0$. Due to convexity we may linearize and obtain

$$f(x) - f(\bar{x}) \geq (\partial_x f(\bar{x}))^\top (x - \bar{x}) \tag{6.56}$$

$$= -\sum_{i=1}^{n} \bar{\alpha}_i \left(\partial_x c_i(\bar{x})\right)^\top (x - \bar{x}) \tag{6.57}$$

$$\geq -\sum_{i=1}^{n} \bar{\alpha}_i (c_i(x) - c_i(\bar{x})) \tag{6.58}$$

$$= -\sum_{i=1}^{n} \bar{\alpha}_i c_i(x) \geq 0. \tag{6.59}$$

Here we used the convexity and differentiability of f to arrive at the rhs of (6.56) and (6.58). To obtain (6.57) we exploited the fact that at the saddle point $\partial_x f(\bar{x})$ can be replaced by the corresponding expansion in $\partial_x c_i(\bar{x})$; thus we used (6.53). Finally, for (6.59) we used the fact that the KKT gap vanishes at the optimum (6.55) and that the constraints are satisfied (6.54). ∎

Optimization by Constraint Satisfaction

In other words, we may solve a convex optimization problem by finding $(\bar{x}, \bar{\alpha})$ that satisfy the conditions of Theorem 6.26. Moreover, these conditions, together with the constraint qualifications of Lemma 6.23, ensure necessity.

Note that we transformed the problem of minimizing functions into one of solving a set of equations, for which several numerical tools are readily available. This is exactly how interior point methods work (see Section 6.4 for details on how to implement them). Necessary conditions on the constraints similar to those discussed previously can also be formulated (see [345] for a detailed discussion).

The other consequence of Theorem 6.26, or rather of the definition of the Lagrangian $L(x, \alpha)$, is that we may bound $f(\bar{x}) = L(\bar{x}, \bar{\alpha})$ from above and below *without* explicit knowledge of $f(\bar{x})$.

Theorem 6.27 (KKT-Gap) *Assume an optimization problem of type (6.37), where both f and c_i are convex and differentiable. Denote by $\bar{x}$ its solution. Then for any set of variables*

(x, α) *with* $\alpha_i \geq 0$, *and for all* $i \in [n]$ *satisfying*

$$\partial_x L(x, \alpha) = 0, \tag{6.60}$$

$$\partial_{\alpha_i} L(x, \alpha) \leq 0 \text{ for all } i \in [n], \tag{6.61}$$

Bounding the Error

we have

$$f(x) \geq f(\bar{x}) \geq f(x) + \sum_{i=1}^{m} \alpha_i c_i(x). \tag{6.62}$$

Strictly speaking, we only need differentiability of f and c_i at $\bar{x}$. However, since $\bar{x}$ is only known *after* the optimization problem has been solved, this is not a very useful condition.

Proof The first part of (6.62) follows from the fact that $x \in X$, so that x satisfies the constraints. Next note that $L(\bar{x}, \bar{\alpha}) = f(\bar{x})$ where $(\bar{x}, \bar{\alpha})$ denotes the saddle point of L. For the second part note that due to the saddle point condition (6.40), we have for any α with $\alpha_i \geq 0$,

$$f(\bar{x}) = L(\bar{x}, \bar{\alpha}) \geq L(\bar{x}, \alpha) \geq \inf_{x' \in X} L(x', \alpha). \tag{6.63}$$

The function $L(x', \alpha)$ is convex in x' since both f' and the constraints c_i are convex and all $\alpha_i \geq 0$. Therefore (6.60) implies that x minimizes $L(x', \alpha)$. This proves the second part of (6.63), which in turn proves the second inequality of (6.62). ∎

Hence, no matter what algorithm we are using in order to solve (6.37), we may always use (6.62) to assess the proximity of the current set of parameters to the solution. Clearly, the relative size of $\sum_{i=1}^{n} \alpha_i c_i(x)$ provides a useful stopping criterion for convex optimization algorithms.

Finally, another concept that is useful when dealing with optimization problems is that of *duality*. This means that for the *primal* minimization problem considered so far, which is expressed in terms of x, we can find a *dual* maximization problem in terms of α by computing the saddle point of the Lagrangian $L(x, \alpha)$, and eliminating the primal variables x. We thus obtain the following dual maximization problem from (6.37):

$$\begin{aligned} &\text{maximize} \quad L(x, \alpha) = f(x) + \sum_{i=1}^{n} \alpha_i c_i(x), \\ &\text{where} \quad (x, \alpha) \in Y := \left\{ (x, \alpha) \;\middle|\; \begin{array}{l} x \in X, \alpha_i \geq 0 \text{ for all } i \in [n] \\ \text{and } \partial_x L(x, \alpha) = 0 \end{array} \right\}. \end{aligned} \tag{6.64}$$

We state without proof a theorem guaranteeing the existence of a solution to (6.64).

Existence of Dual Solution

Theorem 6.28 (Wolfe [607]) *Recall the definition of* X *(6.45) and of the optimization problem (6.37). Under the assumptions that* $\mathcal{X}$ *is an open set,* X *satisfies one of the constraint qualifications of Lemma 6.23, and* f, c_i *are all convex and differentiable, there exists an* $\bar{\alpha} \in \mathbb{R}^n$ *such that* $(\bar{x}, \bar{\alpha})$ *solves the dual optimization problem (6.64) and in addition* $L(\bar{x}, \bar{\alpha}) = f(\bar{x})$.

In order to prove Theorem 6.28 we first have to show that some $(\bar{x}, \bar{\alpha})$ exists satisfying the KKT conditions, and then use the fact that the KKT-Gap at the saddle point vanishes.

6.3.3 Linear and Quadratic Programs

Primal Linear Program

Let us analyze the notions of primal and dual objective functions in more detail by looking at linear and quadratic programs. We begin with a simple linear setting.[6]

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^\top x \\ \text{subject to} \quad & Ax + d \leq 0 \end{aligned} \tag{6.65}$$

where $c, x \in \mathbb{R}^m$, $d \in \mathbb{R}^n$ and $A \in R^{n \times m}$, and where $Ax + d \leq 0$ is a shorthand for $\sum_{j=1}^m A_{ij} x_j + d_i \leq 0$ for all $i \in [n]$.

Unbounded and Infeasible Problems

It is far from clear that (6.65) always has a solution, or indeed a minimum. For instance, the set of x satisfying $Ax + d \leq 0$ might be empty, or it might contain rays going to infinity in directions where $c^\top x$ keeps increasing. Before we deal with this issue in more detail, let us compute the sufficient KKT conditions for optimality, and the dual of (6.65). We may use (6.26) since (6.65) is clearly differentiable and convex. In particular we obtain:

Theorem 6.29 (KKT Conditions for Linear Programs) *A sufficient condition for a solution to the linear program (6.65) to exist is that the following four conditions are satisfied for some $(x, \alpha) \in \mathbb{R}^{m+n}$ where $\alpha \geq 0$:*

$$\partial_x L(x, \alpha) = \partial_x \left[c^\top x + \alpha^\top (Ax + d) \right] = A^\top \alpha + c = 0, \tag{6.66}$$

$$\partial_\alpha L(x, \alpha) = Ax + d \leq 0, \tag{6.67}$$

$$\alpha^\top (Ax + d) = 0, \tag{6.68}$$

$$\alpha \geq 0. \tag{6.69}$$

Then the minimum is given by $c^\top x$.

Note that, depending on the choice of A and d, there may not always exist an x such that $Ax + d \leq 0$, in which case the constraint does not satisfy the conditions of Lemma 6.23. In this situation, no solution exists for (6.65). If a feasible x exists, however, then (projections onto lower dimensional subspaces aside) the constraint qualifications are satisfied on the feasible set, and the conditions above are necessary. See [334, 345, 555] for details.

6. Note that we encounter a small clash of notation in (6.65), since c is used as a symbol for the loss function in the remainder of the book. This inconvenience is outweighed, however, by the advantage of consistency with the standard literature (e.g., [345, 45, 555]) on optimization. The latter will allow the reader to read up on the subject without any need for cumbersome notational changes.

Dual Linear Program

Next we may compute Wolfe's dual optimization problem by substituting (6.66) into $L(x, \alpha)$. Consequently, the primal variables x vanish, and we obtain a maximization problem in terms of α only:

$$\begin{aligned} &\text{maximize} && d^\top \alpha, \\ &\text{subject to} && A^\top \alpha + c = 0 \text{ and } \alpha \geq 0. \end{aligned} \tag{6.70}$$

Note that the number of variables and constraints has changed: we started with m variables and n constraints. Now we have n variables together with m equality constraints and n inequality constraints. While it is not yet completely obvious in the linear case, dualization may render optimization problems more amenable to numerical solution (the contrary may be true as well, though).

Primal Solution $\Leftrightarrow$ Dual Solution

What happens if a solution $\bar{x}$ to the primal problem (6.65) exists? In this case we know (since the KKT conditions of Theorem 6.29 are necessary and sufficient) that there must be an $\bar{\alpha}$ solving the dual problem, since $L(x, \alpha)$ has a saddle point at $(\bar{x}, \bar{\alpha})$.

If no feasible point of the primal problem exists, there must exist, by (a small modification of) Lemma 6.23, some $\alpha \in \mathbb{R}^n$ with $\alpha \geq 0$ and at least one $\alpha_i > 0$ such that $\alpha^\top (Ax + d) > 0$ for all x. This means that for all x, the Lagrangian $L(x, \alpha)$ is unbounded from above, since we can make $\alpha^\top (Ax + d)$ arbitrarily large. Hence the dual optimization problem is unbounded. Using analogous reasoning, if the primal problem is unbounded, the dual problem is infeasible.

Let us see what happens if we dualize (6.70) one more time. First we need more Lagrange multipliers, since we have two sets of constraints. The equality constraints can be taken care of by an unbounded variable x' (see Theorem 6.22 for how to deal with equalities). For the inequalities $\alpha \geq 0$, we introduce a second Lagrange multiplier $y \in \mathbb{R}^n$. After some calculations and resubstitution into the corresponding Lagrangian, we get

$$\begin{aligned} &\text{maximize} && c^\top x', \\ &\text{subject to} && Ax' + d + y = 0 \text{ and } y \geq 0. \end{aligned} \tag{6.71}$$

Dual Dual Linear Program $\rightarrow$ Primal

We can remove $y \geq 0$ from the set of variables by transforming $Ax' + d + y$ into $Ax + d \leq 0$; thus we recover the primal optimization problem (6.65).[7]

The following theorem gives an overview of the transformations and relations between primal and dual problems (see also Table 6.2). Although we only derived these relations for linear programs, they also hold for other convex differentiable settings [45].

Theorem 6.30 (Trichotomy) *For linear and convex quadratic programs exactly one of*

7. This finding is useful if we have to dualize twice in some optimization settings (see Chapter 10), since then we will be able to recover some of the primal variables without further calculations if the optimization algorithm provides us with both primal and dual variables.

Table 6.2 Connections between primal and dual linear and convex quadratic programs.

Primal Optimization Problem (in x)	Dual Optimization Problem (in α)
solution exists	solution exists and extrema are equal
no solution exists	maximization problem has unbounded objective from above or is infeasible
minimization problem has unbounded objective from below or is infeasible	no solution exists
inequality constraint	inequality constraint
equality constraint	free variable
free variable	equality constraint

the following three alternatives must hold:

1. *Both feasible regions are empty.*
2. *Exactly one feasible region is empty, in which case the objective function of the other problem is unbounded in the direction of optimization.*
3. *Both feasible regions are nonempty, in which case both problems have solutions and their extrema are equal.*

Primal Quadratic Program

We conclude this section by stating primal and dual optimization problems, and the sufficient KKT conditions for convex quadratic optimization problems. To keep matters simple we only consider the following type of optimization problem (other problems can be rewritten in the same form; see Problem 6.11 for details):

$$\begin{aligned} &\underset{x}{\text{minimize}} && \tfrac{1}{2}x^\top Kx + c^\top x, \\ &\text{subject to} && Ax + d \leq 0. \end{aligned} \tag{6.72}$$

Here K is a strictly positive definite matrix, $x, c \in \mathbb{R}^m$, $A \in \mathbb{R}^{n\times m}$, and $d \in \mathbb{R}^n$. Note that this is clearly a differentiable convex optimization problem. To introduce a Lagrangian we need corresponding multipliers $\alpha \in \mathbb{R}^n$ with $\alpha \geq 0$. We obtain

$$L(x,\alpha) = \frac{1}{2}x^\top Kx + c^\top x + \alpha^\top(Ax + d). \tag{6.73}$$

Next we may apply Theorem 6.26 to obtain the KKT conditions. They can be stated in analogy to (6.66)–(6.68) as

$$\partial_x L(x,\alpha) = \partial_x \left[c^\top x + \alpha^\top(Ax+d) + \frac{1}{2}x^\top Kx\right] = Kx + A^\top\alpha + c = 0, \tag{6.74}$$

$$\partial_\alpha L(x,\alpha) = Ax + d \leq 0, \tag{6.75}$$

$$\alpha^\top(Ax + d) = 0, \tag{6.76}$$

$$\alpha \geq 0. \tag{6.77}$$

In order to compute the dual of (6.72), we have to eliminate x from (6.73) and write it as a function of α. We obtain

$$L(x,\alpha) = -\frac{1}{2}x^\top Kx + \alpha^\top d \tag{6.78}$$

$$= -\frac{1}{2}\alpha^\top A^\top K^{-1}A\alpha + \left[d - c^\top K^{-1}A^\top\right]\alpha - \frac{1}{2}c^\top K^{-1}c. \tag{6.79}$$

In (6.78) we used (6.74) and (6.76) directly, whereas in order to eliminate x completely in (6.79) we solved (6.74) for $x = -K^{-1}(c + A^\top\alpha)$. Ignoring constant terms this leads to the dual quadratic optimization problem,

Dual Quadratic Program

$$\begin{array}{ll}\underset{\alpha}{\text{minimize}} & -\frac{1}{2}\alpha^\top A^\top K^{-1}A\alpha + \left[d - c^\top K^{-1}A^\top\right]\alpha, \\ \text{subject to} & \alpha \geq 0.\end{array} \tag{6.80}$$

The surprising fact about the dual problem (6.80) is that the constraints become significantly simpler than in the primal (6.72). Furthermore, if $n < m$, we also obtain a more compact representation of the quadratic term.

There is one aspect in which (6.80) differs from its linear counterpart (6.70): if we dualize (6.80) again, we do not recover (6.72) but rather a problem very similar in structure to (6.80). Dualizing (6.80) twice, however, we recover the dual itself (Problem 6.13 deals with this matter in more detail).

6.4 Interior Point Methods

Let us now have a look at simple, yet efficient optimization algorithms for constrained problems: interior point methods.

An interior point is a pair of variables (x, α) that satisfies both primal and dual constraints. As already mentioned before, finding a set of vectors $(\bar{x}, \bar{\alpha})$ that satisfy the KKT conditions is sufficient to obtain a solution in $\bar{x}$. Hence, all we have to do is devise an algorithm which solves (6.74)–(6.77), for instance, if we want to solve a quadratic program. We will focus on the quadratic case — the changes required for linear programs merely involve the removal of some variables, simplifying the equations. See Problem 6.14 and [555, 517] for details.

6.4.1 Sufficient Conditions for a Solution

We need a slight modification of (6.74)–(6.77) in order to achieve our goal: rather than the inequality (6.75), we are better off with an equality and a positivity constraint for an additional variable, i.e. we transform $Ax + d \leq 0$ into $Ax + d + \xi =$

In order to compute the dual of (6.72), we have to eliminate x from (6.73) and write it as a function of α. We obtain

$$L(x,\alpha) = -\frac{1}{2}x^\top Kx + \alpha^\top d \tag{6.78}$$

$$= -\frac{1}{2}\alpha^\top A^\top K^{-1}A\alpha + \left[d - c^\top K^{-1}A^\top\right]\alpha - \frac{1}{2}c^\top K^{-1}c. \tag{6.79}$$

In (6.78) we used (6.74) and (6.76) directly, whereas in order to eliminate x completely in (6.79) we solved (6.74) for $x = -K^{-1}(c + A^\top\alpha)$. Ignoring constant terms this leads to the dual quadratic optimization problem,

Dual Quadratic Program

$$\begin{array}{ll}\underset{\alpha}{\text{minimize}} & -\frac{1}{2}\alpha^\top A^\top K^{-1}A\alpha + \left[d - c^\top K^{-1}A^\top\right]\alpha, \\ \text{subject to} & \alpha \geq 0.\end{array} \tag{6.80}$$

The surprising fact about the dual problem (6.80) is that the constraints become significantly simpler than in the primal (6.72). Furthermore, if $n < m$, we also obtain a more compact representation of the quadratic term.

There is one aspect in which (6.80) differs from its linear counterpart (6.70): if we dualize (6.80) again, we do not recover (6.72) but rather a problem very similar in structure to (6.80). Dualizing (6.80) twice, however, we recover the dual itself (Problem 6.13 deals with this matter in more detail).

6.4 Interior Point Methods

Let us now have a look at simple, yet efficient optimization algorithms for constrained problems: interior point methods.

An interior point is a pair of variables (x, α) that satisfies both primal and dual constraints. As already mentioned before, finding a set of vectors $(\bar{x}, \bar{\alpha})$ that satisfy the KKT conditions is sufficient to obtain a solution in $\bar{x}$. Hence, all we have to do is devise an algorithm which solves (6.74)–(6.77), for instance, if we want to solve a quadratic program. We will focus on the quadratic case — the changes required for linear programs merely involve the removal of some variables, simplifying the equations. See Problem 6.14 and [555, 517] for details.

6.4.1 Sufficient Conditions for a Solution

We need a slight modification of (6.74)–(6.77) in order to achieve our goal: rather than the inequality (6.75), we are better off with an equality and a positivity constraint for an additional variable, i.e. we transform $Ax + d \leq 0$ into $Ax + d + \xi =$

0, where $\xi \geq 0$. Hence we arrive at the following system of equations:

$$\begin{aligned} Kx + A^\top \alpha + c &= 0 \quad \text{(Dual Feasibility)}, \\ Ax + d + \xi &= 0 \quad \text{(Primal Feasibility)}, \\ \alpha^\top \xi &= 0, \\ \alpha, \xi &\geq 0. \end{aligned} \tag{6.81}$$

Optimality as Constraint Satisfaction

Let us analyze the equations in more detail. We have three sets of variables: x, α, ξ. To determine the latter, we have an equal number of equations plus the positivity constraints on α, ξ. While the first two equations are linear and thus amenable to solution, e.g., by matrix inversion, the third equality $\alpha^\top \xi = 0$ has a small defect: given one variable, say α, we cannot solve it for ξ or vice versa. Furthermore, the last two constraints are not very informative either.

We use a primal-dual path-following algorithm, as proposed in [556], to solve this problem. Rather than requiring $\alpha^\top \xi = 0$ we modify it to become $\alpha_i \xi_i = \mu > 0$ for all $i \in [n]$, solve (6.81) for a given μ, and decrease μ to 0 as we go. The advantage of this strategy is that we may use a Newton-type predictor corrector algorithm (see Section 6.2.5) to update the parameters x, α, ξ, which exhibits the fast convergence of a second order method.

6.4.2 Solving the Equations

For the moment, assume that we have suitable initial values of x, α, ξ, and μ with $\alpha, \xi > 0$. Linearization of the first three equations of (6.81), together with the modification $\alpha_i \xi_i = \mu$, yields (we expand x into $x + \Delta x$, etc.):

Linearized Constraints

$$\begin{aligned} K\Delta x + A^\top \Delta\alpha &= -Kx - A^\top \alpha - c &&=: \rho_p, \\ A\Delta x + \Delta\xi &= -Ax - d - \xi &&=: \rho_d, \\ \alpha_i^{-1}\xi_i \Delta\alpha_i + \Delta\xi_i &= \mu\alpha_i^{-1} - \xi_i - \alpha_i^{-1}\Delta\alpha_i \Delta\xi_i &&=: \rho_{\mathrm{KKT}\,i} \text{ for all } i \end{aligned} \tag{6.82}$$

Next we solve for $\Delta\xi_i$ to obtain what is commonly referred to as the *reduced* KKT system. For convenience we use $D := \operatorname{diag}(\alpha_1^{-1}\xi_1, \ldots, \alpha_n^{-1}\xi_n)$ as a shorthand;

$$\begin{bmatrix} K & A^\top \\ A & -D \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\alpha \end{bmatrix} = \begin{bmatrix} \rho_p \\ \rho_d - \rho_{\mathrm{KKT}} \end{bmatrix}. \tag{6.83}$$

We apply a predictor-corrector method as in Section 6.2.5. The resulting matrix of the linear system in (6.83) is indefinite but of full rank, and we can solve (6.83) for $(\Delta x_{\mathrm{Pred}}, \Delta\alpha_{\mathrm{Pred}})$ by explicitly pivoting for individual entries (for instance, solve for Δx first and then substitute the result in to the second equality to obtain $\Delta\alpha$).

This gives us the *predictor* part of the solution. Next we have to correct for the linearization, which is conveniently achieved by updating ρ_{KKT} and solving (6.83) again to obtain the *corrector* values $(\Delta x_{\mathrm{Corr}}, \Delta\alpha_{\mathrm{Corr}})$. The value of $\Delta\xi$ is then obtained from (6.82).

Update in x, α

Next, we have to make sure that the updates in α, ξ do not cause the estimates to violate their positivity constraints. This is done by shrinking the length of $(\Delta x, \Delta\alpha, \Delta\xi)$ by some factor $\lambda \geq 0$, such that

$$\min\left(\frac{\alpha_1 + \lambda\Delta\alpha_1}{\alpha_1}, \ldots, \frac{\alpha_n + \lambda\Delta\alpha_n}{\alpha_n}, \frac{\xi_1 + \lambda\Delta\xi_1}{\xi_1}, \ldots, \frac{\xi_n + \lambda\Delta\xi_n}{\xi_n}\right) \geq \epsilon. \tag{6.84}$$

Of course, only the negative Δ terms pose a problem, since they lead the parameter values closer to 0, which may lead them into conflict with the positivity constraints. Typically [556, 502], we choose $\epsilon = 0.05$. In other words, the solution will not approach the boundaries in α, ξ by more than 95%. See Problem 6.15 for a formula to compute λ.

6.4.3 Updating μ

Next we have to update μ. Here we face the following dilemma: if we decrease μ too quickly, we will get bad convergence of our second order method, since the solution to the problem (which depends on the value of μ) moves too quickly away from our current set of parameters (x, α, ξ). On the other hand, we do not want to spend too much time solving an *approximation* of the unrelaxed ($\mu = 0$) KKT conditions *exactly*. A good indication is how much the positivity constraints would be violated by the current update. Vanderbei [556] proposes the following update of μ:

Tightening the KKT Conditions

$$\mu = \frac{\alpha^\top \xi}{n}\left(\frac{1-\lambda}{10+\lambda}\right)^2. \tag{6.85}$$

The first term gives the average value of satisfaction of the condition $\alpha_i\xi_i = \mu$ after an update step. The second term allows us to decrease μ rapidly if good progress was made (small $(1-\lambda)^2$). Experimental evidence shows that it pays to be slightly more conservative, and to use the *predictor* estimates of α, ξ for (6.85) rather than the corresponding corrector terms.[8] This imposes little overhead for the implementation.

6.4.4 Initial Conditions and Stopping Criterion

To provide a complete algorithm, we have to consider two more things: a stopping criterion and a suitable start value. For the latter, we simply solve a regularized version of the initial reduced KKT system (6.83). This means that we replace K by $K + \mathbf{1}$, use (x, α) in place of $\Delta x, \Delta\alpha$, and replace D by the identity matrix. Moreover, ρ_p and ρ_d are set to the values they would have if all variables had been set to 0 before, and finally ρ_{KKT} is set to 0. In other words, we obtain an initial guess of

Regularized KKT System

8. In practice it is often useful to replace $(1-\lambda)$ by $(1+\epsilon-\lambda)$ for some small $\epsilon > 0$, in order to avoid $\mu = 0$.

(x, α, ξ) by solving

$$\begin{bmatrix} K + \mathbf{1} & A^\top \\ A & -\mathbf{1} \end{bmatrix} \begin{bmatrix} x \\ \alpha \end{bmatrix} = \begin{bmatrix} -c \\ -d \end{bmatrix}, \tag{6.86}$$

and $\xi = -Ax - d$. Since we have to ensure positivity of α, ξ, we simply replace

$$\alpha_i = \max(\alpha_i, 1) \text{ and } \xi_i = \max(\xi_i, 1). \tag{6.87}$$

This heuristic solves the problem of a suitable initial condition.

Regarding the stopping criterion, we recall Theorem 6.27, and in particular (6.62). Rather than obtaining bounds on the precision of *parameters*, we want to make sure that $f(x)$ is close to its optimal value $f(\bar{x})$. From (6.64) we know, provided the feasibility constraints are all satisfied, that the value of the dual objective function is given by $f(x) + \sum_{i=1}^{n} \alpha_i c_i(x)$. We may use the latter to bound the *relative* size of the gap between primal and dual objective function by

$$\text{Gap}(x, \alpha) = \frac{2\left|f(x) - \left(f(x) + \sum_{i=1}^{n} \alpha_i c_i(x)\right)\right|}{|f(x)| + \left|\left(f(x) + \sum_{i=1}^{n} \alpha_i c_i(x)\right)\right|} \leq \frac{-\sum_{i=1}^{n} \alpha_i c_i(x)}{\left|f(x) + \frac{1}{2}\sum_{i=1}^{n} \alpha_i c_i(x)\right|}. \tag{6.88}$$

For the special case where $f(x) = \frac{1}{2}x^\top Kx + c^\top x$ as in (6.72), we know by virtue of (6.73) that the size of the feasibility gap is given by $\alpha^\top \xi$, and therefore

$$\text{Gap}(x, \alpha) = \frac{\alpha^\top \xi}{\left|\frac{1}{2}x^\top Kx + c^\top x + \frac{1}{2}\alpha^\top \xi\right|}. \tag{6.89}$$

In practice, a small number is usually added to the denominator of (6.89) in order to avoid divisions by 0 in the first iteration. The quality of the solution is typically measured on a logarithmic scale by $-\log_{10} \text{Gap}(x, \alpha)$, the number of significant figures.[9] We will come back to specific versions of such interior point algorithms in Chapter 10, and show how Support Vector Regression and Classification problems can be solved with them.

Number of Significant Figures

Primal-Dual path following methods are certainly not the only algorithms that can be employed for minimizing constrained quadratic problems. Other variants, for instance, are Barrier Methods [282, 45, 557], which minimize the unconstrained problem

$$f(x) + \mu \sum_{i=1}^{n} f \ln(-c_i(x)) \text{ for } \mu > 0. \tag{6.90}$$

Active set methods have also been used with success in machine learning [369, 284]. These select subsets of variables x for which the constraints c_i are not ac-

9. Interior point codes are very precise. They usually achieve up to 8 significant figures, whereas iterative approximation methods do not normally exceed more than 3 significant figures on large optimization problems.

tive, i.e., where the we have a strict inequality, and solve the resulting restricted quadratic program, for instance by conjugate gradient descent. We will encounter subset selection methods in Chapter 10.

6.5 Maximum Search Problems

Approximations

In several cases the task of finding an optimal function for estimation purposes means finding the best element from a finite set, or sometimes finding an optimal subset from a finite set of elements. These are discrete (sometimes combinatorial) optimization problems which are not so easily amenable to the techniques presented in the previous two sections. Furthermore, many commonly encountered problems are computationally expensive if solved exactly. Instead, by using probabilistic methods, it is possible to find *almost* optimal approximate solutions. These probabilistic methods are the topic of the present section.

6.5.1 Random Subset Selection

Consider the following problem: given a set of m functions, say $M := \{f_1, \ldots, f_m\}$, and some criterion $Q[f]$, find the function $\hat{f}$ that maximizes $Q[f]$. More formally,

$$\hat{f} := \operatorname*{argmax}_{f \in M} Q[f]. \tag{6.91}$$

Clearly, unless we have additional knowledge about the values $Q[f_i]$, we have to compute all terms $Q[f_i]$ if we want to solve (6.91) exactly. This will cost $O(m)$ operations. If m is large, which is often the case in practical applications, this operation is too expensive. In sparse greedy approximation problems (Section 10.2) or in Kernel Feature Analysis (Section 14.4), m can easily be of the order of 10^5 or larger (here, m is the number of training patterns). Hence we have to look for cheaper *approximate* solutions.

The key idea is to pick a random subset $M' \subset M$ that is sufficiently large, and take the maximum over M' as an approximation of the maximum over M. Provided the distribution of the values of $Q[f_i]$ is "well behaved", i.e., there exists not a small fraction of $Q[f_i]$ whose values are significantly smaller or larger than the average, we will obtain a solution that is close to the optimum with high probability. To formalize these ideas, we need the following result.

Lemma 6.31 (Maximum of Random Variables) *Denote by ξ, ξ' two independent random variables on $\mathbb{R}$ with corresponding distributions $\mathrm{P}_\xi, \mathrm{P}_{\xi'}$ and distribution functions $F_\xi, F_{\xi'}$. Then the random variable $\tilde{\xi} := \max(\xi, \xi')$ has the distribution function $F_{\tilde{\xi}} = F_\xi F_{\xi'}$.*

Proof Note that for a random variable, the distribution function $F(\xi_0)$ is given by

tive, i.e., where the we have a strict inequality, and solve the resulting restricted quadratic program, for instance by conjugate gradient descent. We will encounter subset selection methods in Chapter 10.

6.5 Maximum Search Problems

Approximations

In several cases the task of finding an optimal function for estimation purposes means finding the best element from a finite set, or sometimes finding an optimal subset from a finite set of elements. These are discrete (sometimes combinatorial) optimization problems which are not so easily amenable to the techniques presented in the previous two sections. Furthermore, many commonly encountered problems are computationally expensive if solved exactly. Instead, by using probabilistic methods, it is possible to find *almost* optimal approximate solutions. These probabilistic methods are the topic of the present section.

6.5.1 Random Subset Selection

Consider the following problem: given a set of m functions, say $M := \{f_1, \ldots, f_m\}$, and some criterion $Q[f]$, find the function $\hat{f}$ that maximizes $Q[f]$. More formally,

$$\hat{f} := \operatorname*{argmax}_{f \in M} Q[f]. \tag{6.91}$$

Clearly, unless we have additional knowledge about the values $Q[f_i]$, we have to compute all terms $Q[f_i]$ if we want to solve (6.91) exactly. This will cost $O(m)$ operations. If m is large, which is often the case in practical applications, this operation is too expensive. In sparse greedy approximation problems (Section 10.2) or in Kernel Feature Analysis (Section 14.4), m can easily be of the order of 10^5 or larger (here, m is the number of training patterns). Hence we have to look for cheaper *approximate* solutions.

The key idea is to pick a random subset $M' \subset M$ that is sufficiently large, and take the maximum over M' as an approximation of the maximum over M. Provided the distribution of the values of $Q[f_i]$ is "well behaved", i.e., there exists not a small fraction of $Q[f_i]$ whose values are significantly smaller or larger than the average, we will obtain a solution that is close to the optimum with high probability. To formalize these ideas, we need the following result.

Lemma 6.31 (Maximum of Random Variables) *Denote by ξ, ξ' two independent random variables on $\mathbb{R}$ with corresponding distributions $\mathrm{P}_\xi, \mathrm{P}_{\xi'}$ and distribution functions $F_\xi, F_{\xi'}$. Then the random variable $\tilde{\xi} := \max(\xi, \xi')$ has the distribution function $F_{\tilde{\xi}} = F_\xi F_{\xi'}$.*

Proof Note that for a random variable, the distribution function $F(\xi_0)$ is given by

the probability $\mathrm{P}\{\xi \leq \xi_0\}$. Since ξ and ξ' are independent, we may write

$$F_{\bar{\xi}}(\bar{\xi}) = \mathrm{P}\left\{\max(\xi, \xi') \leq \bar{\xi}\right\} = \mathrm{P}\left\{\xi \leq \bar{\xi} \text{ and } \xi' \leq \bar{\xi}\right\} = \mathrm{P}\left\{\xi \leq \bar{\xi}\right\} \mathrm{P}\left\{\xi' \leq \bar{\xi}\right\} = F_{\xi}(\bar{\xi}) F_{\xi'}(\bar{\xi}), \tag{6.92}$$

which proves the claim. ∎

Distribution Over $\bar{\xi}$ is More Peaked

Repeated application of Lemma 6.31 leads to the following corollary.

Corollary 6.32 (Maximum Over Identical Random Variables) *Let $\xi_1, \ldots, \xi_{\tilde{m}}$ be $\tilde{m}$ independent and identically distributed (iid) random variables, with corresponding distribution function F_{ξ}. Then the random variable $\bar{\xi} := \max(\xi_1, \ldots, \xi_{\tilde{m}})$ has the distribution function $F_{\bar{\xi}}(\bar{\xi}) = \left(F_{\xi}(\bar{\xi})\right)^{\tilde{m}}$.*

In practice, the random variables ξ_i will be the values of $Q[f_i]$, where the f_i are drawn from the set M. If we draw them without replacement (i.e. none of the functions f_i appears twice), however, the values after each draw are dependent and we cannot apply Corollary 6.32 directly. Nonetheless, we can see that the maximum over draws *without* replacement will be larger than the maximum *with* replacement, since recurring observations can be understood as reducing the effective size of the set to be considered. Thus Corollary 6.32 gives us a *lower bound* on the value of the distribution function for draws without replacement. Moreover, for large m the difference between draws with and without replacement is small.

If the distribution of $Q[f_i]$ is known, we may use the distribution directly to determine the size $\tilde{m}$ of a subset to be used to find some $Q[f_i]$ that is almost as good as the solution to (6.91). In all other cases, we have to resort to assessing the *relative* quality of maxima over subsets. The following theorem tells us how.

Best Element of a Subset

Theorem 6.33 (Ranks on Random Subsets) *Denote by $M := \{x_1, \ldots, x_m\} \subset \mathbb{R}$ a set of cardinality m, and by $\tilde{M} \subset M$ a random subset of size $\tilde{m}$. Then the probability that* $\max \tilde{M}$ *is greater equal than n elements of M is at least $1 - \left(\frac{n}{m}\right)^{\tilde{m}}$.*

Proof We prove this by assuming the converse, namely that $\max \tilde{M}$ is smaller than $(m - n)$ elements of M. For $\tilde{m} = 1$ we know that this probability is $\frac{n}{m}$, since there are n elements to choose from. For $\tilde{m} > 1$, the probability is the one of choosing $\tilde{m}$ elements out of a subset M_{low} of n elements, rather than all m elements. Therefore we have that

$$\mathrm{P}(\tilde{M} \subset M_{\text{low}}) = \frac{\binom{n}{\tilde{m}}}{\binom{m}{\tilde{m}}} = \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \ldots \cdot \frac{n-\tilde{m}+1}{m-\tilde{m}+1} < \left(\frac{n}{m}\right)^{\tilde{m}}.$$

Consequently the probability that the maximum over $\tilde{M}$ will be larger than n elements of M is given by $1 - \mathrm{P}(\tilde{M} \subset M_{\text{low}}) \geq 1 - \left(\frac{n}{m}\right)^{\tilde{m}}$. ∎

The practical consequence is that we may use $1 - \left(\frac{n}{m}\right)^{\tilde{m}}$ to compute the required size of a random subset to achieve the desired degree of approximation. If we want to obtain results in the $\frac{n}{m}$ percentile range with $1 - \eta$ confidence, we must

solve for $\tilde{m} = \frac{\log(1-\eta)}{\ln n/m}$. To give a numerical example, if we desire values that are better than 95% of all other estimates with $1 - 0.05$ probability, then $\kappa = 59$ samples are sufficient. This $(95\%, 95\%, 59)$ rule is very useful in practice.[10] A similar method was used to speed up the process of boosting classifiers in the MadaBoost algorithm [143]. Furthermore, one could think whether it might not be useful to recycle old observations rather than computing all 59 values from scratch. If this can be done cheaply, and under some additional independence assumptions, subset selection methods can be improved further. For details see [424] who use the method in the context of memory management for operating systems.

6.5.2 Random Evaluation

Quite often, the evaluation of the term $Q[f]$ itself is rather time consuming, especially if $Q[f]$ is the sum of many (m, for instance) iid random variables. Again, we can speed up matters considerably by using probabilistic methods. The key idea is that averages over independent random variables are concentrated, which is to say that averages over subsets do not differ too much from averages over the whole set.

Approximating Sums by Partial Sums

Hoeffding's Theorem (Section 5.2) quantifies the size of the deviations between the expectation of a sum of random variables and their values at individual trials. We will use this to bound deviations between averages over sets and subsets. All we have to do is translate Theorem 5.1 into a statement regarding sample averages over different sample sizes. This can be readily constructed as follows:

Corollary 6.34 (Deviation Bounds for Empirical Means [508]) *Suppose $\xi_1, \ldots, \xi_m$ are iid bounded random variables, falling into the interval $[a, a+b]$ with probability one. Denote their average by $Q_m = \frac{1}{m}\sum_i \xi_i$. Furthermore, denote by $\xi_{s(1)}, \ldots, \xi_{s(\tilde{m})}$ with $\tilde{m} < m$ a subset of the same random variables (with $s : \{1, \ldots, \tilde{m}\} \to \{1, \ldots, m\}$ being an injective map, i.e. $s(i) = s(j)$ only if $i = j$), and $Q_{\tilde{m}} = \frac{1}{\tilde{m}}\sum_i \xi_{s(i)}$. Then for any $\varepsilon > 0$,*

Deviation of Subsets

$$\left.\begin{array}{l} \mathrm{P}\{Q_m - Q_{\tilde{m}} \geq \varepsilon\} \\ \mathrm{P}\{Q_{\tilde{m}} - Q_m \geq \varepsilon\} \end{array}\right\} \leq \exp\left(-\frac{2m\tilde{m}\varepsilon^2}{(m-\tilde{m})b^2}\right) = \exp\left(-2m\frac{\varepsilon^2}{b^2}\frac{\frac{\tilde{m}}{m}}{1-\frac{\tilde{m}}{m}}\right) \tag{6.93}$$

Proof By construction $\mathbf{E}[Q_m - Q_{\tilde{m}}] = 0$, since Q_m and $Q_{\tilde{m}}$ are both averages over sums of random variables drawn from the same distribution. Hence we only have to rewrite $Q_m - Q_{\tilde{m}}$ as an average over (different) random variables to apply Hoeffding's bound. Since all Q_i are identically distributed, we may pick the first $\tilde{m}$ random variables, without loss of generality. In other words, we assume that

10. During World War I tanks were often numbered in continuous increasing order. Unfortunately this "feature" allowed the enemy to estimate the number of tanks. How?

$s(i) = i$ for $i = 1, \ldots, \tilde{m}$. Then

$$Q_m - Q_{\tilde{m}} = \frac{1}{m}\sum_{i=1}^{m} \xi_i - \frac{1}{\tilde{m}}\sum_{i=1}^{\tilde{m}} \xi_i = \frac{1}{m}\sum_{i=1}^{\tilde{m}} \left(1 - \frac{m}{\tilde{m}}\right)\xi_i + \frac{1}{m}\sum_{i=\tilde{m}+1}^{m} \xi_i. \tag{6.94}$$

Thus we may split up $Q_m - Q_{\tilde{m}}$ into a sum of $\tilde{m}$ random variables with range $b_i = (\frac{m}{\tilde{m}} - 1)b$, and $m - \tilde{m}$ random variables with range $b_i = b$. We obtain

$$\sum_{i=1}^{m} b_i^2 = b^2\tilde{m}\left(\frac{m}{\tilde{m}} - 1\right)^2 + (m - \tilde{m})b^2 = b^2(m - \tilde{m})\frac{m}{\tilde{m}}. \tag{6.95}$$

Substituting this into (5.7) and noting that $Q_m - Q_{\tilde{m}} - \mathbf{E}[Q_m - Q_{\tilde{m}}] = Q_m - Q_{\tilde{m}}$ completes the proof. ∎

For small $\frac{\tilde{m}}{m}$ the rhs in (6.93) reduces to $\exp\left(-\frac{2\tilde{m}\varepsilon^2}{b^2}\right)$. In other words, deviations on the subsample $\tilde{m}$ dominate the overall deviation of $Q_m - Q_{\tilde{m}}$ from 0. This allows us to compute a cutoff criterion for evaluating Q_m by computing only a subset of its terms.

Cutoff Criterion

We need only solve (6.93) for $\frac{\tilde{m}}{m}$. Hence, in order to ensure that $Q_{\tilde{m}}$ is within ε of Q_m with probability $1 - \eta$, we have to take a fraction $\frac{\tilde{m}}{m}$ of samples that satisfies

$$\frac{\frac{\tilde{m}}{m}}{1 - \frac{\tilde{m}}{m}} = \frac{b^2(\ln 2 - \ln \eta)}{2m\varepsilon^2} =: c, \text{ and therefore } \frac{\tilde{m}}{m} = \frac{c}{1 + c}. \tag{6.96}$$

The fraction $\frac{\tilde{m}}{m}$ can be small for large m, which is exactly the case where we need methods to speed up evaluation.

6.5.3 Greedy Optimization Strategies

Quite often the overall goal is not necessarily to find the single best element x_i from a set X to solve a problem, but to find a good subset $\tilde{X} \subset X$ of size $\tilde{m}$ according to some quality criterion $Q[\tilde{X}]$. Problems of this type include approximating a matrix by a subset of its rows and columns (Section 10.2), finding approximate solutions to Kernel Fisher Discriminant Analysis (Chapter 15) and finding a sparse solution to the problem of Gaussian Process Regression (Section 16.3.4). These all have a common structure:

Applications

(i) Finding an optimal set $\tilde{X} \subset X$ is quite often a combinatorial problem, or it even may be NP-hard, since it means selecting $\tilde{m} = |\tilde{X}|$ elements from a set of $m = |X|$ elements. There are $\binom{m}{\tilde{m}}$ different choices, which clearly prevents an exhaustive search over all of them. Additionally, the size of $\tilde{m}$ is often not known beforehand. Hence we need a fast approximate algorithm.

(ii) The evaluation of $Q[\tilde{X} \cup \{x_i\}]$ is inexpensive, provided $Q[\tilde{X}]$ has been computed before. This indicates that an iterative algorithm can be useful.

(iii) The value of $Q[X]$, or equivalently how well we would do by taking the whole set X, can be bounded efficiently by using $Q[\tilde{X}]$ (or some by-products of the computation of $Q[\tilde{M}]$) without actually computing $Q[X]$.

Algorithm 6.6 Sparse Greedy Algorithm

Require: Set of functions X, Precision ϵ, Criterion $Q[\cdot]$
  Set $\tilde{X} = \emptyset$
  repeat
    Choose random subset X' of size m' from $X \backslash \tilde{X}$.
    Pick $\hat{x} = \text{argmax}_{x \in X'} Q[X' \cup \{x\}]$
    $X' = X' \cup \{\hat{x}\}$
    If needed, (re)compute bound on $Q[X]$.
  until $Q[\tilde{X}] + \epsilon \geq$ Bound on $Q[X]$
Output: $\tilde{X}, Q[\tilde{X}]$

(iv) The set of functions X is typically very large (i.e. more than 10^5 elements), yet the individual improvements by f_i via $Q[\tilde{X} \cup \{x_i\}]$ do not differ too much, meaning that specific x_i for which $Q[\tilde{X} \cup \{x_i\}]$ deviate by a large amount from the rest of $Q[\tilde{X} \cup \{x_i\}]$ do not exist.

Iterative Enlargement of $\tilde{X}$

In this case we may use a sparse greedy algorithm to find near optimal solutions among the remaining $X \backslash \tilde{X}$ elements. This combines the idea of an iterative enlargement of $\tilde{X}$ by one more element at a time (which is feasible since we can compute $Q[\tilde{X} \cup \{f_i\}]$ cheaply) with the idea that we need not consider all f_i as possible candidates for the enlargement. This uses the reasoning in Section 6.5.1 combined with the fact that the distribution of the improvements is not too long tailed (cf. (iv)). The overall strategy is described in Algorithm 6.6.

Problems 6.9 and 6.10 contain more examples of sparse greedy algorithms.

6.6 Summary

This chapter gave an overview of different optimization methods, which form the basic toolbox for solving the problems arising in learning with kernels. The main focus was on convex and differentiable problems, hence the overview of properties of convex sets and functions defined on them.

The key insights in Section 6.1 are that *convex sets* can be defined by *level sets of convex functions* and that convex optimization problems have *one global minimum*. Furthermore, the fact that the solutions of convex maximization over polyhedral sets can be found on the vertices will prove useful in some unsupervised learning applications (Section 14.4).

Basic tools for unconstrained problems (Section 6.2) include interval cutting methods, the Newton method, Conjugate Gradient descent, and Predictor-Corrector methods. These techniques are often used as building blocks to solve more advanced constrained optimization problems.

Since constrained minimization is a fairly complex topic, we only presented a selection of fundamental results, such as necessary and sufficient conditions in the general case of nonlinear programming. The KKT conditions for differentiable

Algorithm 6.6 Sparse Greedy Algorithm

Require: Set of functions X, Precision ϵ, Criterion $Q[\cdot]$
  Set $\tilde{X} = \emptyset$
  repeat
    Choose random subset X' of size m' from $X \backslash \tilde{X}$.
    Pick $\hat{x} = \text{argmax}_{x \in X'} Q[X' \cup \{x\}]$
    $X' = X' \cup \{\hat{x}\}$
    If needed, (re)compute bound on $Q[X]$.
  until $Q[\tilde{X}] + \epsilon \geq$ Bound on $Q[X]$
Output: $\tilde{X}, Q[\tilde{X}]$

(iv) The set of functions X is typically very large (i.e. more than 10^5 elements), yet the individual improvements by f_i via $Q[\tilde{X} \cup \{x_i\}]$ do not differ too much, meaning that specific x_i for which $Q[\tilde{X} \cup \{x_i\}]$ deviate by a large amount from the rest of $Q[\tilde{X} \cup \{x_i\}]$ do not exist.

Iterative Enlargement of $\tilde{X}$

In this case we may use a sparse greedy algorithm to find near optimal solutions among the remaining $X \backslash \tilde{X}$ elements. This combines the idea of an iterative enlargement of $\tilde{X}$ by one more element at a time (which is feasible since we can compute $Q[\tilde{X} \cup \{f_i\}]$ cheaply) with the idea that we need not consider all f_i as possible candidates for the enlargement. This uses the reasoning in Section 6.5.1 combined with the fact that the distribution of the improvements is not too long tailed (cf. (iv)). The overall strategy is described in Algorithm 6.6.

Problems 6.9 and 6.10 contain more examples of sparse greedy algorithms.

6.6 Summary

This chapter gave an overview of different optimization methods, which form the basic toolbox for solving the problems arising in learning with kernels. The main focus was on convex and differentiable problems, hence the overview of properties of convex sets and functions defined on them.

The key insights in Section 6.1 are that *convex sets* can be defined by *level sets of convex functions* and that convex optimization problems have *one global minimum*. Furthermore, the fact that the solutions of convex maximization over polyhedral sets can be found on the vertices will prove useful in some unsupervised learning applications (Section 14.4).

Basic tools for unconstrained problems (Section 6.2) include interval cutting methods, the Newton method, Conjugate Gradient descent, and Predictor-Corrector methods. These techniques are often used as building blocks to solve more advanced constrained optimization problems.

Since constrained minimization is a fairly complex topic, we only presented a selection of fundamental results, such as necessary and sufficient conditions in the general case of nonlinear programming. The KKT conditions for differentiable

convex functions then followed immediately from the previous reasoning. The main results are dualization, meaning the transformation of optimization problems via the Lagrangian mechanism into possibly simpler problems, and that optimality properties can be estimated via the KKT gap (Theorem 6.27).

Interior point algorithms are practical applications of the duality reasoning; these seek to find a solution to optimization problems by satisfying the KKT optimality conditions. Here we were able to employ some of the concepts introduced at an earlier stage, such as predictor corrector methods and numerical ways of finding roots of equations. These algorithms are robust tools to find solutions on moderately sized problems ($10^3 - 10^4$ examples). Larger problems require decomposition methods, to be discussed in Section 10.4, or randomized methods. The chapter concluded with an overview of randomized methods for maximizing functions or finding the best subset of elements. These techniques are useful once datasets are so large that we cannot reasonably hope to find exact solutions to optimization problems.

6.7 Problems

6.1 (Level Sets •) *Given the function $f : \mathbb{R}^2 \to \mathbb{R}$ with $f(x) := |x_1|^p + |x_2|^p$, for which p do we obtain a convex function?*

Now consider the sets $\{x|f(x) \leq c\}$ for some $c > 0$. Can you give an explicit parametrization of the boundary of the set? Is it easier to deal with this parametrization? Can you find other examples (see also [489] and Chapter 8 for details)?

6.2 (Convex Hulls •) *Show that for any set X, its convex hull* $\operatorname{co} X$ *is convex. Furthermore, show that* $\operatorname{co} X = X$ *if X is convex.*

6.3 (Method of False Position [334] •••) *Given a unimodal (possessing one minimum) differentiable function $f : \mathbb{R} \to \mathbb{R}$, develop a quadratic method for minimizing f.*

Hint: Recall the Newton method. There we used $f''(x)$ to make a quadratic approximation of f. Two values of $f'(x)$ are also sufficient to obtain this information, however.

What happens if we may only use f? What does the iteration scheme look like? See Figure 6.8 for a hint.

6.4 (Convex Minimization in one Variable ••) *Denote by f a convex function on $[a, b]$. Show that the algorithm below finds the minimum of f. What is the* rate *of convergence in x to* $\operatorname{argmin}_x f(x)$*? Can you obtain a bound in $f(x)$ wrt.* $\min_x f(x)$*?*

input *a, b, f and threshold ε*
  $x_1 = a, x_2 = \frac{a+b}{2}, x_3 = b$ and compute $f(x_1), f(x_2), f(x_3)$
  repeat
    if *$x_3 - x_2 > x_2 - x_1$* ***then***

convex functions then followed immediately from the previous reasoning. The main results are dualization, meaning the transformation of optimization problems via the Lagrangian mechanism into possibly simpler problems, and that optimality properties can be estimated via the KKT gap (Theorem 6.27).

Interior point algorithms are practical applications of the duality reasoning; these seek to find a solution to optimization problems by satisfying the KKT optimality conditions. Here we were able to employ some of the concepts introduced at an earlier stage, such as predictor corrector methods and numerical ways of finding roots of equations. These algorithms are robust tools to find solutions on moderately sized problems ($10^3 - 10^4$ examples). Larger problems require decomposition methods, to be discussed in Section 10.4, or randomized methods. The chapter concluded with an overview of randomized methods for maximizing functions or finding the best subset of elements. These techniques are useful once datasets are so large that we cannot reasonably hope to find exact solutions to optimization problems.

6.7 Problems

6.1 (Level Sets •) *Given the function $f : \mathbb{R}^2 \to \mathbb{R}$ with $f(x) := |x_1|^p + |x_2|^p$, for which p do we obtain a convex function?*

Now consider the sets $\{x|f(x) \leq c\}$ for some $c > 0$. Can you give an explicit parametrization of the boundary of the set? Is it easier to deal with this parametrization? Can you find other examples (see also [489] and Chapter 8 for details)?

6.2 (Convex Hulls •) *Show that for any set X, its convex hull* $\operatorname{co} X$ *is convex. Furthermore, show that* $\operatorname{co} X = X$ *if X is convex.*

6.3 (Method of False Position [334] •••) *Given a unimodal (possessing one minimum) differentiable function $f : \mathbb{R} \to \mathbb{R}$, develop a quadratic method for minimizing f.*

Hint: Recall the Newton method. There we used $f''(x)$ to make a quadratic approximation of f. Two values of $f'(x)$ are also sufficient to obtain this information, however.

What happens if we may only use f? What does the iteration scheme look like? See Figure 6.8 for a hint.

6.4 (Convex Minimization in one Variable ••) *Denote by f a convex function on $[a, b]$. Show that the algorithm below finds the minimum of f. What is the* rate *of convergence in x to* $\operatorname{argmin}_x f(x)$*? Can you obtain a bound in $f(x)$ wrt.* $\min_x f(x)$*?*

input *a, b, f and threshold ε*
  $x_1 = a, x_2 = \frac{a+b}{2}, x_3 = b$ and compute $f(x_1), f(x_2), f(x_3)$
  repeat
    if *$x_3 - x_2 > x_2 - x_1$* ***then***

$x_4 = \frac{x_2+x_3}{2}$ *and compute* $f(x_4)$
else
$x_4 = \frac{x_1+x_2}{2}$ *and compute* $f(x_4)$
end if
Keep the two points closest to the point with the minimum value of $f(x_i)$ *and rename them such that* $x_1 < x_2 < x_3$.
until $x_3 - x_1 \geq \varepsilon$

6.5 (Newton Method in $\mathbb{R}^d$ ••) *Extend the Newton method to functions on* $\mathbb{R}^d$. *What does the iteration rule look like? Under which conditions does the algorithm converge? Do you have to extend Theorem 6.13 to prove convergence?*

6.6 (Rewriting Quadratic Functionals •) *Given a function*

$$f(x) = x^\top Q x + c^\top x + d, \tag{6.97}$$

rewrite it into the form of (6.18). Give explicit expressions for $x^* = \operatorname{argmin}_x f(x)$ *and the difference in the additive constants.*

6.7 (Kantorovich Inequality [278] •••) *Prove Theorem 6.16. Hint: note that without loss of generality we may require* $\|x\|^2 = 1$. *Second, perform a transformation of coordinates into the eigensystem of K. Finally, note that in the new coordinate system we are dealing with convex combinations of eigenvalues* λ_i *and* $\frac{1}{\lambda_i}$. *First show (6.24) for only two eigenvalues. Then argue that only the largest and smallest eigenvalues matter.*

6.8 (Random Subsets •) *Generate m random numbers drawn uniformly from the interval* $[0, 1]$. *Plot their distribution function. Plot the distribution of maxima of subsets of random numbers. What can you say about the distribution of the maxima? What happens if you draw randomly from the Laplace distribution, with density* $p(\xi) = e^{-\xi}$ *(for* $\xi \geq 0$*)?*

6.9 (Matching Pursuit [342] ••) *Denote by* $f_1, \ldots, f_M$ *a set of functions* $\mathcal{X} \to \mathbb{R}$, *by* $\{x_1, \ldots, x_m\} \subset \mathcal{X}$ *a set of locations and by* $\{y_1, \ldots, y_m\} \subset \mathcal{Y}$ *a set of corresponding observations.*

Design a sparse greedy algorithm that finds a linear combination of functions $f := \sum_i \alpha_i f_i$ *minimizing the squared loss between* $f(x_i)$ *and* y_i.

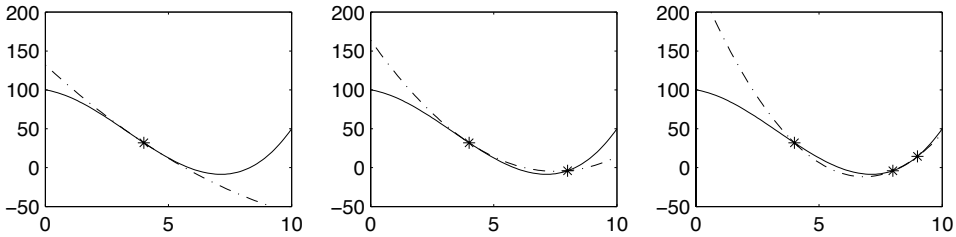


Figure 6.8 From left to right: Newton method, method of false position, quadratic interpolation through 3 points. Solid line: $f(x)$, dash-dotted line: interpolation.

6.10 (Reduced Set Approximation [474] ••) *Let $f(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x)$ be a kernel expansion in a Reproducing Kernel Hilbert Space $\mathcal{H}_k$ (see Section 2.2.3). Give a sparse greedy algorithm that finds an approximation to f in $\mathcal{H}_k$ by using fewer terms. See also Chapter 18 for more detail.*

6.11 (Equality Constraints in LP and QP ••) *Find the dual optimization problem and the necessary KKT conditions for the following optimization problem:*

$$\begin{array}{ll} \underset{x}{\text{minimize}} & c^\top x, \\ \textit{subject to} & Ax + b \leq 0, \\ & Cx + d = 0, \end{array} \tag{6.98}$$

where $c, x \in \mathbb{R}^m$, $b \in \mathbb{R}^n$, $d \in \mathbb{R}^{n'}$, $A \in R^{n \times m}$ and $C \in \mathbb{R}^{n'}$. Hint: split up the equality constraints into two inequality constraints. Note that you may combine the two Lagrange multipliers again to obtain a free variable. Derive the corresponding conditions for

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \frac{1}{2} x^\top K x + c^\top x, \\ \textit{subject to} & Ax + b \leq 0, \\ & Cx + d = 0, \end{array} \tag{6.99}$$

where K is a strictly positive definite matrix.

6.12 (Not Strictly Definite Quadratic Parts •••) *How do you have to change the dual of (6.99) if K does not have full rank? Is it better not to dualize in this case? Do the KKT conditions still hold?*

6.13 (Dual Problems of Quadratic Programs ••) *Denote by P a quadratic optimization problem of type (6.72) and by $(\cdot)^D$ the dualization operation. Prove that the following is true,*

$$((P^D)^D)^D = P^D \text{ and } (((P^D)^D)^D)^D = (P^D)^D, \tag{6.100}$$

where in general $(P^D)^D \neq P$. Hint: use (6.80). Caution: you have to check whether $KA^\top$ has full rank.

6.14 (Interior Point Equations for Linear Programs [336] •••) *Derive the interior point equations for linear programs. Hint: use the expansions for the quadratic programs and note that the reduced KKT system has only a diagonal term where we had K before.*

How does the complexity of the problem scale with the size of A?

6.15 (Update Step in Interior Point Codes •) *Show that the maximum value of λ satisfying (6.84) can be found by*

$$\frac{1}{\lambda} = \max\left(1, (\epsilon - 1)^{-1} \min_{i \in [n]} \frac{\Delta\alpha_i}{\alpha_i}, (\epsilon - 1)^{-1} \min_{i \in [n]} \frac{\Delta\xi_i}{\xi_i}\right). \tag{6.101}$$

we cannot give an exhaustive listing of all successful SVM applications. We thus conclude the list with some of the more exotic applications, such as in High-Energy-Physics [19, 558], in the monitoring of household appliances [390], in protein secondary structure prediction [249], and, with rather intriguing results, in the design of decision feedback equalizers (DFE) in telephony [105].

7.9 Summary

This chapter introduced SV pattern recognition algorithms. The crucial idea is to use kernels to reduce a complex classification task to one that can be solved with separating hyperplanes. We discussed what kind of hyperplane should be constructed in order to get good generalization performance, leading to the idea of large margins. It turns out that the concept of large margins can be justified in a number of different ways, including arguments based on statistical learning theory, and compression schemes. We described in detail how the optimal margin hyperplane can be obtained as the solution of a quadratic programming problem. We started with the linear case, where the hyperplane is constructed in the space of the inputs, and then moved on to the case where we use a kernel function to compute dot products, in order to compute the hyperplane in a feature space.

Two further extensions greatly increase the applicability of the approach. First, to deal with noisy data, we introduced so-called slack variables in the optimization problem. Second, for problems that have more than just two classes, we described a number of generalizations of the binary SV classifiers described initially.

Finally, we reported applications and benchmark comparisons for the widely used USPS handwritten digit task. SVMs turn out to work very well in this field, as well as in a variety of other domains mentioned briefly.

7.10 Problems

7.1 (Weight Vector Scaling •) *Show that instead of the "*1*" on the right hand side of the separation constraint (7.11), we can use any positive number* $\gamma > 0$*, without changing the optimal margin hyperplane solution. What changes in the soft margin case?*

7.2 (Dual Perceptron Algorithm [175] ••) *Kernelize the perceptron algorithm described in footnote 1. Which of the patterns will appear in the expansion of the solution?*

7.3 (Margin of Optimal Margin Hyperplanes [62] ••) *Prove that the geometric margin* ρ *of the optimal margin hyperplane can be computed from the solution* $\boldsymbol{\alpha}$ *via*

$$\rho^{-2} = \sum_{i=1}^{m} \alpha_i. \tag{7.68}$$

Also prove that

$$\rho^{-2} = 2W(\boldsymbol{\alpha}) = \|\mathbf{w}\|^2. \tag{7.69}$$

Note that for these relations to hold true, $\boldsymbol{\alpha}$ needs to be the solution of (7.29).

7.4 (Relationship Between $\|\mathbf{w}\|$ and the Geometrical Margin •) *(i) Consider a separating hyperplane in canonical form. Prove that the margin, measured perpendicularly to the hyperplane, equals $1/\|\mathbf{w}\|$, by considering two opposite points which precisely satisfy $|\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1$.*

(ii) How does the corresponding statement look for the case of ν-SVC? Use the constraint (7.41), and assume that all slack variables are 0.

7.5 (Compression Bound for Large Margin Classification ○○○) *Formalize the ideas stated in Section 7.2: Assuming that the data are separable and lie in a ball of radius R, how many bits are necessary to encode the labels of the data by encoding the parameters of a hyperplane? Formulate a generalization error bound in terms of the compression ratio by using the analysis of Vapnik [561, Section 4.6]. Compare the resulting bound with Theorem 7.3. Take into account the eigenvalues of the Gram matrix, using the ideas of from [604] (cf. Section 12.4).*

7.6 (Positive Definiteness of the SVC Hessian •) *From Definition 2.4, prove that the matrix $Q_{ij} := (y_i y_j k(x_i, x_j))_{ij}$ is positive definite.*

7.7 (Geometric Interpretation of Duality in SVC [38] ••) *Prove that the programming problem (7.10), (7.11) has the same solution as (7.22), provided the threshold b is adjusted such that the hyperplane bisects the shortest connection of the two convex hulls. Hint: Show that the latter is the dual of the former. Interpret the result geometrically.*

7.8 (Number of Points Required to Define a Hyperplane •) *From (7.22), argue that no matter what the dimensionality of the space, there can always be situations where two training points suffice to determine the optimal hyperplane.*

7.9 (In-Bound SVs in Soft Margin SVMs •) *Prove that in-bound SVs lie exactly on the margin. Hint: Use the KKT conditions, and proceed analogously to Section 7.3, where it was shown that in the hard margin case, all SVs lie exactly on the margin.*

Argue, moreover, that bound SVs can lie both on or in the margin, and that they will "usually" lie in the margin.

7.10 (Pattern-Dependent Regularization •) *Derive a version of the soft margin classification algorithm which uses different regularization constants C_i for each training example. Start from (7.35), replace the second term by $\frac{1}{m}\sum_{i=1}^{m} C_i \xi_i$, and derive the dual. Discuss both the mathematical form of the result, and possible applications (cf. [462]).*

7.11 (Uncertain Labels ••) *In this chapter, we have been concerned mainly with the case where the patterns are assigned to one of two classes, i.e., $y \in \{\pm 1\}$. Consider now the*

case where the assignment is not strict, i.e., $y \in [-1, 1]$. Modify the soft margin variants of the SV algorithm, (7.34), (7.35) and (7.41), (7.40), such that

- *whenever $y = 0$, the corresponding pattern has effectively no influence*
- *if all labels are in $\{\pm 1\}$, the original algorithm is recovered*
- *if $|y| < 1$, then the corresponding pattern has less influence than it would have for $|y| = 1$.*

7.12 (SVMs vs. Parzen Windows ○○○) *Develop algorithms that approximate the SVM (soft or hard margin) solution by starting from the Parzen Windows algorithm (Figure 1.1) and sparsifying the expansion of the solution.*

7.13 (Squared Error SVC [111] ●●) *Derive a version of the soft margin classification algorithm which penalizes the errors quadratically. Start from (7.35), replace the second term by $\frac{1}{m}\sum_{i=1}^{m}\xi_i^2$, and derive the dual. Compare the result to the usual C-SVM, both in terms of algorithmic differences and in terms of robustness properties. Which algorithm would you expect to work better for Gaussian-like noise, which one for noise with longer tails (and thus more outliers) (cf. Chapter 3)?*

7.14 (C-SVC with Group Error Penalty ●●) *Suppose the training data are partitioned into ℓ groups,*

$$\begin{array}{c} (\mathbf{x}_1^1, y_1^1), \ldots, (\mathbf{x}_1^{m_1}, y_1^{m_1}) \\ \vdots \qquad\qquad \vdots \\ (\mathbf{x}_\ell^1, y_\ell^1), \ldots, (\mathbf{x}_\ell^{m_\ell}, y_\ell^{m_\ell}), \end{array} \tag{7.70}$$

where $\mathbf{x}_i^j \in \mathcal{H}$ and $y_i^j \in \{\pm 1\}$ (it is understood that the index i runs over $\{1, 2, \ldots, \ell\}$ and the index j runs over $\{1, 2, \ldots, m_i\}$).

Suppose, moreover, that we would like to count a point as misclassified already if one point belonging to the same group is misclassified.

Design an SV algorithm where each group's penalty equals the slack of the worst point in that group.

Hint: Use the objective function

$$\frac{1}{2}\|\mathbf{w}\|^2 + \sum_i C_i \xi_i, \tag{7.71}$$

and the constraints

$$y_i^j \cdot (\left\langle \mathbf{w}, \mathbf{x}_i^j \right\rangle + b) \geq 1 - \xi_i, \tag{7.72}$$

$$\xi_i \geq 0. \tag{7.73}$$

Show that the dual problem consists of maximizing

$$W(\boldsymbol{\alpha}) = \sum_{i,j} \alpha_i^j - \frac{1}{2} \sum_{i,j,i',j'} \alpha_i^j \alpha_{i'}^{j'} y_i^j y_{i'}^{j'} \left\langle \mathbf{x}_i^j, \mathbf{x}_{i'}^{j'} \right\rangle, \tag{7.74}$$

subject to

$$0 = \sum_{i,j} \alpha_i^j y_i^j,\ 0 \le \alpha_i^j,\ \textit{and}\ \sum_j \alpha_i^j \le C_i. \tag{7.75}$$

Argue that typically, only one point per group will become an SV.
Show that C-SVC is a special case of this algorithm.

7.15 (ν-SVC with Group Error Penalty •••) *Derive a ν-version of the algorithm in Problem 7.14.*

7.16 (C-SVC vs. ν-SVC ••) *As a modification of ν-SVC (Section 7.5), compute the dual of $\tau(\mathbf{w}, \boldsymbol{\xi}, \rho) = \|\mathbf{w}\|^2/2 + C(-\nu\rho + (1/m)\sum_{i=1}^m \xi_i)$ (note that in ν-SVC, $C = 1$ is used). Argue that due to the homogeneity of the objective function, the dual solution gets scaled by C, however, the decision function will not change. Hence we may set $C = 1$.*

7.17 (Multi-class vs. Binary SVC [593] ••) *(i) Prove that the multi-class SVC formulation of (7.59) specializes to the binary C-SVC (7.35) in the case $k = 2$, by using $\mathbf{w}_1 = -\mathbf{w}_2$, $b_1 = -b_2$, and $\xi_i = \frac{1}{2}\xi_i^r$ for pattern $\mathbf{x}_i$ in class r. (ii) Derive the dual of (7.59).*

7.18 (Multi-Class ν-SVC ○○○) *Derive a ν-version of the approach described in Section 7.6.4.*

7.19 (LPM with Constrained Signs •) *Modify the LPM algorithm such that it is guaranteed that each expansion pattern will have a coefficient v_i whose sign equals the class label y_i. Hint: Do not introduce additional constraints, but eliminate the α_i^* variables and use a different ansatz for the solution.*

7.20 (Multi-Class LPM [593] ••) *In analogy to Section 7.6.4, develop a multi-class version of the LP machine (Section 7.7).*

7.21 (Version Space [368, 239, 451, 238] •••) *Consider hyperplanes passing through the origin, $\{\mathbf{x} | \langle \mathbf{w}, \mathbf{x} \rangle = 0\}$, with weight vectors $\mathbf{w} \in \mathcal{H}$, $\|\mathbf{w}\| = 1$. The set of all such hyperplanes forms a unit sphere in weight space. Each training example $(\mathbf{x}, y) \in \mathcal{H} \times \{\pm 1\}$ splits the sphere into two halves: one that correctly classifies $(\mathbf{x}, y)$, i.e., $\operatorname{sgn} \langle \mathbf{w}, \mathbf{x} \rangle = y$, and one that does not. Each training example thus corresponds to a hemisphere (or, equivalently, an oriented great circle) in weight space, and a training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ corresponds to the intersection of m hemispheres, called the* version space.

1. *Discuss how the distances between the training example and the hyperplane in the two representations are related.*
2. *Discuss the relationship to the idea of the Hough transform [255]. The Hough transform is sometimes used in image processing to detect lines. In a nutshell, each point gets to cast votes in support for all potential lines that are consistent with it, and at the end, the lines can be read off the histogram of votes.*
3. *Prove that if all $\mathbf{x}_i$ have the same norm, the maximum margin weight vector corresponds to the center of the largest $m - 1$-dimensional sphere that fits into version space.*

4. *Construct situations where the center of the above largest sphere will generalize poorly, and compare it to the center of mass of version space, called the* Bayes point.

5. *If you disregard the labels of the training examples, there is no longer a single area on the unit sphere which is distinguished from the others due to its corresponding to the correct labelling. Instead, the sphere is split into a number of cells. Argue that the expectation of the natural logarithm of this number equals the VC entropy (Section 5.5.6).*

7.22 (Kernels on Sets ○○○) *Use the construction of Proposition 2.19 to define a kernel that compares two points $\mathbf{x}, \mathbf{x}' \in \mathcal{H}$ by comparing the version spaces (see Problem 7.21) of the labelled examples $(\mathbf{x}, 1)$ and $(\mathbf{x}', 1)$. Define a prior distribution P on the unit sphere in $\mathcal{H}$, and discuss the implications of its choice for the induced kernel. What can you say about the connection between this kernel and the kernel $\langle \mathbf{x}, \mathbf{x}' \rangle$?*

7.23 (Training Algorithms for ν-SVC ○○○) *Try to come up with efficient training algorithms for ν-SVC, building on the material presented in Chapter 10.*

(i) Design a simple chunking algorithm that gradually removes all non-SVs.

(ii) Design a decomposition algorithm.

(iii) Is it possible to modify the SMO algorithm such that it deals with the additional equality constraint that ν-SVC comes with? What is the smallest set of patterns that you can optimize over without violating the two equality constraints? Can you design a generalized SMO algorithm for this case?

7.24 (Prior Class Probabilities ●●) *Suppose that it is known a priori that π_+ and π_- are the probabilities that a pattern belongs to the class ± 1, respectively. Discuss ways of modifying the simple classification algorithm described in Section 1.2 to take this information into account.*

7.25 (Choosing C ●●) *Suppose that R is a measure for the range of the data in feature space that scales like the length of the points in $\mathcal{H}$ (cf. Section 7.8.1). Argue that C should scale like $1/R^2$.*[15] *Hint: consider scaling the data by some $\gamma > 0$. How do you have to scale C such that $f(x) = \langle \mathbf{w}, \Phi(x_j) \rangle + b$ (where $\mathbf{w} = \sum_i \alpha_i y_i \Phi(x_i)$) remains invariant ($j \in [m]$)?*[16] *Discuss measures R that can be used. Why does $R := \max_j k(x_j, x_j)$ not make sense for the Gaussian RBF kernel?*

Moreover, argue that in the asymptotic regime, the upper bound on the α_j should scale with $1/m$, justifying the use of m in (7.38).

7.26 (Choosing C, Part II ○○○) *Problem 7.25 does not take into account the class labels, and hence also not the potential overlap of the two classes. Note that this is different in the ν-approach, which automatically scales the margin with the noise. Can you modify the recommendation in Problem 7.25 to get a selection criterion for C which takes into account the labels, e.g., in the form of prior information on the noise level?*

15. Thanks to Olivier Chapelle for this suggestion.
16. Note that in the ν-parametrization, this scale invariance comes for free.

7 Pattern Recognition

This chapter is devoted to a detailed description of SV classification (SVC) methods. We have already briefly visited the SVC algorithm in Chapter 1. There will be some overlap with that chapter, but here we give a more thorough treatment.

Overview

We start by describing the classifier that forms the basis for SVC, the separating hyperplane (Section 7.1). Separating hyperplanes can differ in how large a margin of separation they induce between the classes, with corresponding consequences on the generalization error, as discussed in Section 7.2. The "optimal" margin hyperplane is defined in Section 7.3, along with a description of how to compute it. Using the kernel trick of Chapter 2, we generalize to the case where the optimal margin hyperplane is not computed in input space, but in a feature space nonlinearly related to the latter (Section 7.4). This dramatically increases the applicability of the approach, as does the introduction of slack variables to deal with outliers and noise in the data (Section 7.5). Many practical problems require us to classify the data into more than just two classes. Section 7.6 describes how multi-class SV classification systems can be built. Following this, Section 7.7 describes some variations on standard SV classification algorithms, differing in the regularizers and constraints that are used. We conclude with a fairly detailed section on experiments and applications (Section 7.8).

Prerequisites

This chapter requires basic knowledge of kernels, as conveyed in the first half of Chapter 2. To understand details of the optimization problems, it is helpful (but not indispensable) to get some background from Chapter 6. To understand the connections to learning theory, in particular regarding the statistical basis of the regularizer used in SV classification, it would be useful to have read Chapter 5.

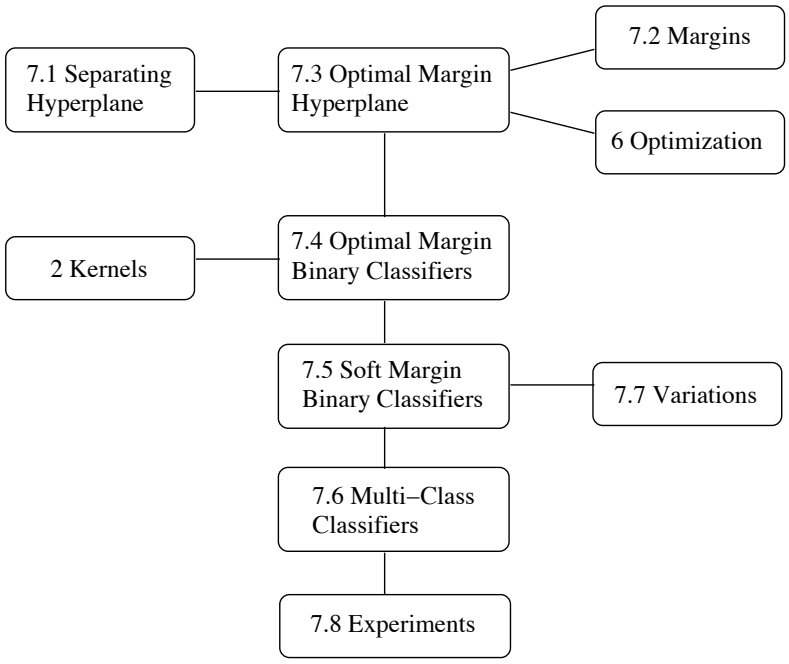
7.1 Separating Hyperplanes

Hyperplane

Suppose we are given a dot product space $\mathcal{H}$, and a set of pattern vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathcal{H}$. Any hyperplane in $\mathcal{H}$ can be written as

$$\{\mathbf{x} \in \mathcal{H} \mid \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}, \quad \mathbf{w} \in \mathcal{H}, b \in \mathbb{R}. \tag{7.1}$$

In this formulation, $\mathbf{w}$ is a vector orthogonal to the hyperplane: If $\mathbf{w}$ has unit length, then $\langle \mathbf{w}, \mathbf{x} \rangle$ is the length of $\mathbf{x}$ along the direction of $\mathbf{w}$ (Figure 7.1). For general $\mathbf{w}$, this number will be scaled by $\|\mathbf{w}\|$. In any case, the set (7.1) consists



of vectors that all have the same length along $\mathbf{w}$. In other words, these are vectors that project onto the same point on the line spanned by $\mathbf{w}$.

In this formulation, we still have the freedom to multiply $\mathbf{w}$ and b by the same non-zero constant. This superfluous freedom — physicists would call it a "gauge" freedom — can be abolished as follows.

Definition 7.1 (Canonical Hyperplane) *The pair* $(\mathbf{w}, b) \in \mathcal{H} \times \mathbb{R}$ *is called a* canonical *form of the hyperplane (7.1) with respect to* $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathcal{H}$*, if it is scaled such that*

$$\min_{i=1,\ldots,m} |\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1, \tag{7.2}$$

which amounts to saying that the point closest to the hyperplane has a distance of $1/\|\mathbf{w}\|$ *(Figure 7.2).*

Note that the condition (7.2) still allows two such pairs: given a canonical hyperplane $(\mathbf{w}, b)$, another one satisfying (7.2) is given by $(-\mathbf{w}, -b)$. For the purpose of pattern recognition, these two hyperplanes turn out to be different, as they are oriented differently; they correspond to two *decision functions*,

Decision Function

$$\begin{aligned} f_{\mathbf{w},b} : \mathcal{H} &\to \{\pm 1\} \\ \mathbf{x} &\mapsto f_{\mathbf{w},b}(\mathbf{x}) = \operatorname{sgn}\left(\langle \mathbf{w}, \mathbf{x} \rangle + b\right), \end{aligned} \tag{7.3}$$

which are the inverse of each other.

In the absence of class labels $y_i \in \{\pm 1\}$ associated with the $\mathbf{x}_i$, there is no way of distinguishing the two hyperplanes. For a *labelled* dataset, a distinction exists: The two hyperplanes make opposite class assignments. In pattern recognition,

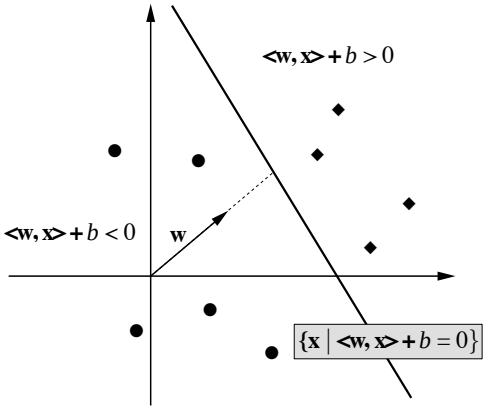


Figure 7.1 A separable classification problem, along with a separating hyperplane, written in terms of an orthogonal weight vector $\mathbf{w}$ and a threshold b. Note that by multiplying both $\mathbf{w}$ and b by the same non-zero constant, we obtain the same hyperplane, represented in terms of different parameters. Figure 7.2 shows how to eliminate this scaling freedom.

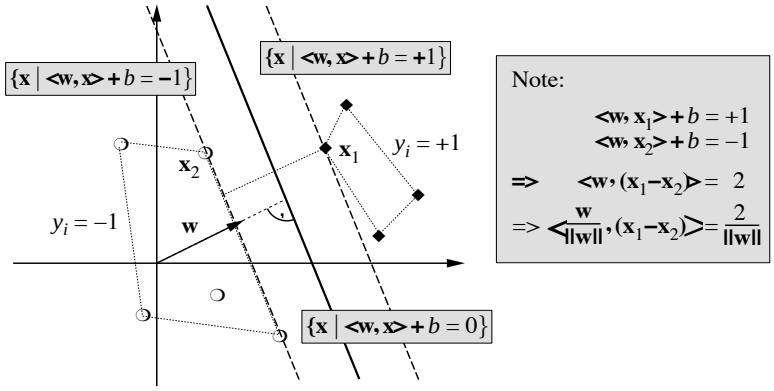


Figure 7.2 By requiring the scaling of $\mathbf{w}$ and b to be such that the point(s) closest to the hyperplane satisfy $|\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1$, we obtain a *canonical* form $(\mathbf{w}, b)$ of a hyperplane. Note that in this case, the margin, measured perpendicularly to the hyperplane, equals $1/\|\mathbf{w}\|$. This can be seen by considering two opposite points which precisely satisfy $|\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1$ (cf. Problem 7.4)

we attempt to find a solution $f_{\mathbf{w},b}$ which *correctly classifies* the labelled examples $(\mathbf{x}_i, y_i) \in \mathcal{H} \times \{\pm 1\}$; in other words, which satisfies $f_{\mathbf{w},b}(\mathbf{x}_i) = y_i$ for all i (in this case, the training set is said to be *separable*), or at least for a large fraction thereof.

The next section will introduce the term *margin*, to denote the distance to a separating hyperplane from the point closest to it. It will be argued that to generalize well, a large margin should be sought. In view of Figure 7.2, this can be achieved by keeping $\|\mathbf{w}\|$ small. Readers who are content with this level of detail may skip the next section and proceed directly to Section 7.3, where we describe how to construct the hyperplane with the largest margin.

7.2 The Role of the Margin

The margin plays a crucial role in the design of SV learning algorithms. Let us start by formally defining it.

Definition 7.2 (Geometrical Margin) *For a hyperplane* $\{\mathbf{x} \in \mathcal{H} | \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}$*, we call*

$$\rho_{(\mathbf{w},b)}(\mathbf{x}, y) := y(\langle \mathbf{w}, \mathbf{x} \rangle + b)/\|\mathbf{w}\| \tag{7.4}$$

Geometrical Margin

the geometrical margin of the point $(\mathbf{x}, y) \in \mathcal{H} \times \{\pm 1\}$. *The minimum value*

$$\rho_{(\mathbf{w},b)} := \min_{i=1,\ldots,m} \rho_{(\mathbf{w},b)}(\mathbf{x}_i, y_i) \tag{7.5}$$

shall be called the geometrical margin of $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$*. If the latter is omitted, it is understood that the training set is meant.*

Occasionally, we will omit the qualification *geometrical*, and simply refer to the *margin*.

For a point $(\mathbf{x}, y)$ which is correctly classified, the margin is simply the distance from $\mathbf{x}$ to the hyperplane. To see this, note first that the margin is zero *on* the hyperplane. Second, in the definition, we effectively consider a hyperplane

$$(\hat{\mathbf{w}}, \hat{b}) := (\mathbf{w}/\|\mathbf{w}\|, b/\|\mathbf{w}\|), \tag{7.6}$$

which has a unit length weight vector, and then compute the quantity $y(\langle \hat{\mathbf{w}}, \mathbf{x} \rangle + \hat{b})$. The term $\langle \hat{\mathbf{w}}, \mathbf{x} \rangle$, however, simply computes the length of the projection of $\mathbf{x}$ onto the direction orthogonal to the hyperplane, which, after adding the offset $\hat{b}$, equals the distance to it. The multiplication by y ensures that the margin is positive whenever a point is correctly classified. For misclassified points, we thus get a margin which equals the *negative* distance to the hyperplane. Finally, note that for canonical hyperplanes, the margin is $1/\|\mathbf{w}\|$ (Figure 7.2). The definition of the canonical hyperplane thus ensures that the length of $\mathbf{w}$ now corresponds to a meaningful geometrical quantity.

Margin of Canonical Hyperplanes

It turns out that the margin of a separating hyperplane, and thus the length of the weight vector $\mathbf{w}$, plays a fundamental role in support vector type algorithms. Loosely speaking, if we manage to separate the training data with a large margin, then we have reason to believe that we will do well on the test set. Not surprisingly, there exist a number of explanations for this intuition, ranging from the simple to the rather technical. We will now briefly sketch some of them.

Insensitivity to Pattern Noise

The simplest possible justification for large margins is as follows. Since the training and test data are assumed to have been generated by the same underlying dependence, it seems reasonable to assume that most of the test patterns will lie close (in $\mathcal{H}$) to at least one of the training patterns. For the sake of simplicity, let us consider the case where *all* test points are generated by adding bounded pattern noise (sometimes called input noise) to the training patterns. More precisely, given a training point $(\mathbf{x}, y)$, we will generate test points of the form $(\mathbf{x} + \Delta\mathbf{x}, y)$, where

7.2 The Role of the Margin

The margin plays a crucial role in the design of SV learning algorithms. Let us start by formally defining it.

Definition 7.2 (Geometrical Margin) *For a hyperplane* $\{\mathbf{x} \in \mathcal{H} | \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}$*, we call*

$$\rho_{(\mathbf{w},b)}(\mathbf{x}, y) := y(\langle \mathbf{w}, \mathbf{x} \rangle + b)/\|\mathbf{w}\| \tag{7.4}$$

Geometrical Margin

the geometrical margin of the point $(\mathbf{x}, y) \in \mathcal{H} \times \{\pm 1\}$. *The minimum value*

$$\rho_{(\mathbf{w},b)} := \min_{i=1,\ldots,m} \rho_{(\mathbf{w},b)}(\mathbf{x}_i, y_i) \tag{7.5}$$

shall be called the geometrical margin of $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$*. If the latter is omitted, it is understood that the training set is meant.*

Occasionally, we will omit the qualification *geometrical*, and simply refer to the *margin*.

For a point $(\mathbf{x}, y)$ which is correctly classified, the margin is simply the distance from $\mathbf{x}$ to the hyperplane. To see this, note first that the margin is zero *on* the hyperplane. Second, in the definition, we effectively consider a hyperplane

$$(\hat{\mathbf{w}}, \hat{b}) := (\mathbf{w}/\|\mathbf{w}\|, b/\|\mathbf{w}\|), \tag{7.6}$$

which has a unit length weight vector, and then compute the quantity $y(\langle \hat{\mathbf{w}}, \mathbf{x} \rangle + \hat{b})$. The term $\langle \hat{\mathbf{w}}, \mathbf{x} \rangle$, however, simply computes the length of the projection of $\mathbf{x}$ onto the direction orthogonal to the hyperplane, which, after adding the offset $\hat{b}$, equals the distance to it. The multiplication by y ensures that the margin is positive whenever a point is correctly classified. For misclassified points, we thus get a margin which equals the *negative* distance to the hyperplane. Finally, note that for canonical hyperplanes, the margin is $1/\|\mathbf{w}\|$ (Figure 7.2). The definition of the canonical hyperplane thus ensures that the length of $\mathbf{w}$ now corresponds to a meaningful geometrical quantity.

Margin of Canonical Hyperplanes

It turns out that the margin of a separating hyperplane, and thus the length of the weight vector $\mathbf{w}$, plays a fundamental role in support vector type algorithms. Loosely speaking, if we manage to separate the training data with a large margin, then we have reason to believe that we will do well on the test set. Not surprisingly, there exist a number of explanations for this intuition, ranging from the simple to the rather technical. We will now briefly sketch some of them.

Insensitivity to Pattern Noise

The simplest possible justification for large margins is as follows. Since the training and test data are assumed to have been generated by the same underlying dependence, it seems reasonable to assume that most of the test patterns will lie close (in $\mathcal{H}$) to at least one of the training patterns. For the sake of simplicity, let us consider the case where *all* test points are generated by adding bounded pattern noise (sometimes called input noise) to the training patterns. More precisely, given a training point $(\mathbf{x}, y)$, we will generate test points of the form $(\mathbf{x} + \Delta\mathbf{x}, y)$, where

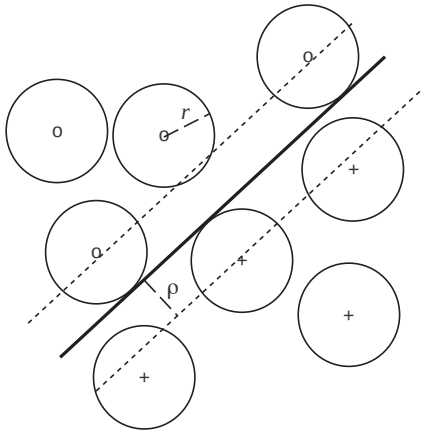


Figure 7.3 Two-dimensional toy example of a classification problem: Separate 'o' from '+' using a hyperplane. Suppose that we add bounded noise to each pattern. If the optimal margin hyperplane has margin ρ, and the noise is bounded by $r < \rho$, then the hyperplane will correctly separate even the noisy patterns. Conversely, if we ran the perceptron algorithm (which finds *some* separating hyperplane, but not necessarily the optimal one) on the noisy data, then we would recover the optimal hyperplane in the limit $r \to \rho$.

$\Delta\mathbf{x} \in \mathcal{H}$ is bounded in norm by some $r > 0$. Clearly, if we manage to separate the training set with a margin $\rho > r$, we will correctly classify *all* test points: Since all training points have a distance of at least ρ to the hyperplane, the test patterns will still be on the correct side (Figure 7.3, cf. also [152]).

If we knew ρ beforehand, then this could actually be turned into an optimal margin classifier training algorithm, as follows. If we use an r which is slightly smaller than ρ, then even the patterns with added noise will be separable with a nonzero margin. In this case, the standard perceptron algorithm can be shown to converge.[1]

Therefore, we can run the perceptron algorithm on the noisy patterns. If the algorithm finds a sufficient number of noisy versions of each pattern, with different perturbations $\Delta\mathbf{x}$, then the resulting hyperplane will not intersect any of the balls depicted in Figure 7.3. As r approaches ρ, the resulting hyperplane should better approximate the maximum margin solution (the figure depicts the limit $r = \rho$). This constitutes a connection between training with pattern noise and maximizing the margin. The latter, in turn, can be thought of as a regularizer, comparable to those discussed earlier (see Chapter 4 and (2.49)). Similar connections to training with noise, for other types of regularizers, have been pointed out before for neural networks [50].

1. Rosenblatt's perceptron algorithm [439] is one of the simplest conceivable iterative procedures for computing a separating hyperplane. In its simplest form, it proceeds as follows. We start with an arbitrary weight vector $\mathbf{w}_0$. At step $n \in \mathbb{N}$, we consider the training example $(\mathbf{x}_n, y_n)$. If it is classified correctly using the current weight vector (i.e., if $\operatorname{sgn}\langle \mathbf{x}_n, \mathbf{w}_{n-1}\rangle = y_n$), we set $\mathbf{w}_n := \mathbf{w}_{n-1}$; otherwise, we set $\mathbf{w}_n := \mathbf{w}_{n-1} + \eta y_i \mathbf{x}_i$ (here, $\eta > 0$ is a learning rate). We thus loop over all patterns repeatedly, until we can complete one full pass through the training set without a single error. The resulting weight vector will thus classify all points correctly. Novikoff [386] proved that this procedure terminates, provided that the training set is separable with a nonzero margin.

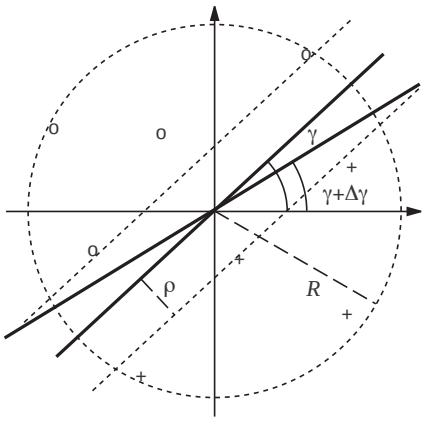


Figure 7.4 Two-dimensional toy example of a classification problem: Separate 'o' from '+' using a hyperplane passing through the origin. Suppose the patterns are bounded in length (distance to the origin) by R, and the classes are separated by an optimal hyperplane (parametrized by the angle γ) with margin ρ. In this case, we can perturb the parameter by some $\Delta\gamma$ with $|\Delta\gamma| < \arcsin \frac{\rho}{R}$, and still correctly separate the data.

Parameter Noise

A similar robustness argument can be made for the dependence of the hyperplane on the parameters $(\mathbf{w}, b)$ (cf. [504]). If all points lie at a distance of at least ρ from the hyperplane, and the patterns are bounded in length, then small perturbations to the hyperplane parameters will not change the classification of the training data (see Figure 7.4).[2] Being able to perturb the parameters of the hyperplane amounts to saying that to store the hyperplane, we need fewer bits than we would for a hyperplane whose *exact* parameter settings are crucial. Interestingly, this is related to what is called the Minimum Description Length principle ([583, 433, 485], cf. also [522, 305, 94]): The best description of the data, in terms of generalization error, should be the one that requires the fewest bits to store.

VC Margin Bound

We now move on to a more technical justification of large margin algorithms. For simplicity, we only deal with hyperplanes that have offset $b = 0$, leaving $f(\mathbf{x}) = \operatorname{sgn} \langle \mathbf{w}, \mathbf{x} \rangle$. The theorem below follows from a result in [24].

Theorem 7.3 (Margin Error Bound) *Consider the set of decision functions* $f(\mathbf{x}) = \operatorname{sgn} \langle \mathbf{w}, \mathbf{x} \rangle$ *with* $\|\mathbf{w}\| \leq \Lambda$ *and* $\|\mathbf{x}\| \leq R$, *for some* $R, \Lambda > 0$. *Moreover, let* $\rho > 0$, *and* ν *denote the fraction of training examples with margin smaller than* $\rho / \|\mathbf{w}\|$, *referred to as the* margin error.

Margin Error

For all distributions P *generating the data, with probability at least* $1 - \delta$ *over the drawing of the* m *training patterns, and for any* $\rho > 0$ *and* $\delta \in (0, 1)$, *the probability that a test pattern drawn from* P *will be misclassified is bounded from above, by*

$$\nu + \sqrt{\frac{c}{m} \left(\frac{R^2 \Lambda^2}{\rho^2} \ln^2 m + \ln(1/\delta) \right)}. \tag{7.7}$$

Here, c *is a universal constant.*

2. Note that this would not hold true if we allowed patterns of arbitrary length — this type of restriction of the pattern lengths pops up in various places, such as Novikoff's theorem [386], Vapnik's VC dimension bound for margin classifiers (Theorem 5.5), and Theorem 7.3.

Let us try to understand this theorem. It makes a probabilistic statement about a probability, by giving an upper bound on the probability of test error, which *itself* only holds true with a certain probability, $1-\delta$. Where do these two probabilities come from? The first is due to the fact that the *test* examples are randomly drawn from P; the second is due to the *training* examples being drawn from P. Strictly speaking, the bound does not refer to a *single* classifier that has been trained on some fixed data set at hand, but to an ensemble of classifiers, trained on various instantiations of training sets generated by the same underlying regularity P.

It is beyond the scope of the present chapter to prove this result. The basic ingredients of bounds of this type, commonly referred to as *VC bounds*, are described in Chapter 5; for further details, see Chapter 12, and [562, 491, 504, 125]. Several aspects of the bound are noteworthy. The test error is bounded by a sum of the margin error ν, and a capacity term (the $\sqrt{\ldots}$ term in (7.7)), with the latter tending to zero as the number of examples, m, tends to infinity. The capacity term can be kept small by keeping R and Λ small, and making ρ large. If we assume that R and Λ are fixed a priori, the main influence is ρ. As can be seen from (7.7), a large ρ leads to a small capacity term, but the margin error ν gets larger. A small ρ, on the other hand, will usually cause fewer points to have margins smaller than $\rho/\|\mathbf{w}\|$, leading to a smaller margin error; but the capacity penalty will increase correspondingly. The overall message: Try to find a hyperplane which is aligned such that even for a large ρ, there are few margin errors.

Maximizing ρ, however, is the same as minimizing the length of $\mathbf{w}$. Hence we might just as well keep ρ fixed, say, equal to 1 (which is the case for canonical hyperplanes), and search for a hyperplane which has a small $\|\mathbf{w}\|$ and few points with a margin smaller than $1/\|\mathbf{w}\|$; in other words (Definition 7.2), few points such that $y\,\langle \mathbf{w}, \mathbf{x}\rangle < 1$.

It should be emphasized that dropping the condition $\|\mathbf{w}\| \le \Lambda$ would prevent us from stating a bound of the kind shown above. We could give an alternative bound, where the capacity depends on the dimensionality of the space $\mathcal{H}$. The crucial advantage of the bound given above is that it is independent of that dimensionality, enabling us to work in very high dimensional spaces. This will become important when we make use of the kernel trick.

It has recently been pointed out that the margin also plays a crucial role in improving asymptotic rates in nonparametric estimation [551]. This topic, however, is beyond the scope of the present book.

Implementation in Hardware

To conclude this section, we note that large margin classifiers also have advantages of a practical nature: An algorithm that can separate a dataset with a certain margin will behave in a benign way when implemented in hardware. Real-world systems typically work only within certain accuracy bounds, and if the classifier is insensitive to small changes in the inputs, it will usually tolerate those inaccuracies.

We have thus accumulated a fair amount of evidence in favor of the following approach: Keep the margin training error small, and the margin large, in order to achieve high generalization ability. In other words, hyperplane decision functions

should be constructed such that they maximize the margin, and at the same time separate the training data with as few exceptions as possible. Sections 7.3 and 7.5 respectively will deal with these two issues.

7.3 Optimal Margin Hyperplanes

Let us now derive the optimization problem to be solved for computing the optimal hyperplane. Suppose we are given a set of examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, $\mathbf{x}_i \in \mathcal{H}$, $y_i \in \{\pm 1\}$. Here and below, the index i runs over $1, \ldots, m$ by default. We assume that there is at least one negative and one positive y_i. We want to find a decision function $f_{\mathbf{w},b}(\mathbf{x}) = \operatorname{sgn}\left(\langle \mathbf{w}, \mathbf{x}\rangle + b\right)$ satisfying

$$f_{\mathbf{w},b}(\mathbf{x}_i) = y_i. \tag{7.8}$$

If such a function exists (the non-separable case will be dealt with later), canonicality (7.2) implies

$$y_i \left(\langle \mathbf{x}_i, \mathbf{w}\rangle + b\right) \geq 1. \tag{7.9}$$

As an aside, note that out of the two canonical forms of the same hyperplane, $(\mathbf{w}, b)$ and $(-\mathbf{w}, -b)$, only one will satisfy equations (7.8) and (7.11). The existence of class labels thus allows to distinguish two orientations of a hyperplane.

Following the previous section, a separating hyperplane which generalizes well can thus be constructed by solving the following problem:

$$\underset{\mathbf{w}\in\mathcal{H}, b\in\mathbb{R}}{\text{minimize}} \quad \tau(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2, \tag{7.10}$$

$$\text{subject to} \quad y_i \left(\langle \mathbf{x}_i, \mathbf{w}\rangle + b\right) \geq 1 \text{ for all } i = 1, \ldots, m. \tag{7.11}$$

This is called the *primal optimization problem*.

Problems like this one are the subject of optimization theory. For details on how to solve them, see Chapter 6; for a short intuitive explanation, cf. the remarks following (1.26) in the introductory chapter. We will now derive the so-called *dual problem*, which can be shown to have the same solutions as (7.10). In the present case, it will turn out that it is more convenient to deal with the dual. To derive it, we introduce the Lagrangian,

Lagrangian

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i \left(y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) - 1\right), \tag{7.12}$$

with Lagrange multipliers $\alpha_i \geq 0$. Recall that as in Chapter 1, we use bold face Greek variables to refer to the corresponding vectors of variables, for instance, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$.

The Lagrangian L must be maximized with respect to α_i, and minimized with respect to $\mathbf{w}$ and b (see Theorem 6.26). Consequently, at this saddle point, the

should be constructed such that they maximize the margin, and at the same time separate the training data with as few exceptions as possible. Sections 7.3 and 7.5 respectively will deal with these two issues.

7.3 Optimal Margin Hyperplanes

Let us now derive the optimization problem to be solved for computing the optimal hyperplane. Suppose we are given a set of examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$, $\mathbf{x}_i \in \mathcal{H}$, $y_i \in \{\pm 1\}$. Here and below, the index i runs over $1, \ldots, m$ by default. We assume that there is at least one negative and one positive y_i. We want to find a decision function $f_{\mathbf{w},b}(\mathbf{x}) = \operatorname{sgn}\left(\langle \mathbf{w}, \mathbf{x}\rangle + b\right)$ satisfying

$$f_{\mathbf{w},b}(\mathbf{x}_i) = y_i. \tag{7.8}$$

If such a function exists (the non-separable case will be dealt with later), canonicality (7.2) implies

$$y_i \left(\langle \mathbf{x}_i, \mathbf{w}\rangle + b\right) \geq 1. \tag{7.9}$$

As an aside, note that out of the two canonical forms of the same hyperplane, $(\mathbf{w}, b)$ and $(-\mathbf{w}, -b)$, only one will satisfy equations (7.8) and (7.11). The existence of class labels thus allows to distinguish two orientations of a hyperplane.

Following the previous section, a separating hyperplane which generalizes well can thus be constructed by solving the following problem:

$$\underset{\mathbf{w}\in\mathcal{H}, b\in\mathbb{R}}{\text{minimize}} \quad \tau(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2, \tag{7.10}$$

$$\text{subject to} \quad y_i \left(\langle \mathbf{x}_i, \mathbf{w}\rangle + b\right) \geq 1 \text{ for all } i = 1, \ldots, m. \tag{7.11}$$

This is called the *primal optimization problem*.

Problems like this one are the subject of optimization theory. For details on how to solve them, see Chapter 6; for a short intuitive explanation, cf. the remarks following (1.26) in the introductory chapter. We will now derive the so-called *dual problem*, which can be shown to have the same solutions as (7.10). In the present case, it will turn out that it is more convenient to deal with the dual. To derive it, we introduce the Lagrangian,

Lagrangian

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i \left(y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) - 1\right), \tag{7.12}$$

with Lagrange multipliers $\alpha_i \geq 0$. Recall that as in Chapter 1, we use bold face Greek variables to refer to the corresponding vectors of variables, for instance, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)$.

The Lagrangian L must be maximized with respect to α_i, and minimized with respect to $\mathbf{w}$ and b (see Theorem 6.26). Consequently, at this saddle point, the

derivatives of L with respect to the primal variables must vanish,

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \quad \frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \tag{7.13}$$

which leads to

$$\sum_{i=1}^{m} \alpha_i y_i = 0, \tag{7.14}$$

and

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i. \tag{7.15}$$

The solution vector thus has an expansion in terms of training examples. Note that although the solution $\mathbf{w}$ is unique (due to the strict convexity of (7.10), and the convexity of (7.11)), the coefficients α_i need not be.

According to the KKT theorem (Chapter 6), only the Lagrange multipliers α_i that are non-zero at the saddle point, correspond to constraints (7.11) which are precisely met. Formally, for all $i = 1, \ldots, m$, we have

$$\alpha_i [y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1] = 0. \tag{7.16}$$

Support Vectors

The patterns $\mathbf{x}_i$ for which $\alpha_i > 0$ are called *Support Vectors*. This terminology is related to corresponding terms in the theory of convex sets, relevant to convex optimization (e.g., [334, 45]).[3] According to (7.16), they lie exactly on the margin.[4] All remaining examples in the training set are irrelevant: Their constraints (7.11) are satisfied automatically, and they do not appear in the expansion (7.15), since their multipliers satisfy $\alpha_i = 0$.[5]

This leads directly to an upper bound on the generalization ability of optimal margin hyperplanes. To this end, we consider the so-called leave-one-out method (for further details, see Section 12.2) to estimate the expected test error [335, 559]. This procedure is based on the idea that if we leave out one of the training

3. Given any boundary point of a convex set, there always exists a hyperplane separating the point from the interior of the set. This is called a *supporting hyperplane*.

SVs lie on the boundary of the convex hulls of the two classes, thus they possess supporting hyperplanes. The SV optimal hyperplane is the hyperplane which lies in the middle of the two parallel supporting hyperplanes (of the two classes) with maximum distance.

Conversely, from the optimal hyperplane, we can obtain supporting hyperplanes for all SVs of both classes, by shifting it by $1/\|\mathbf{w}\|$ in both directions.

4. Note that this implies the solution $(\mathbf{w}, b)$, where b is computed using $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) = 1$ for SVs, *is* in canonical form with respect to the training data. (This makes use of the reasonable assumption that the training set contains both positive and negative examples.)

5. In a statistical mechanics framework, Anlauf and Biehl [12] have put forward a similar argument for the *optimal stability perceptron*, also computed using constrained optimization. There is a large body of work in the physics community on optimal margin classification. Some further references of interest are [310, 191, 192, 394, 449, 141]; other early works include [313].

examples, and train on the remaining ones, then the probability of error on the left out example gives us a fair indication of the true test error. Of course, doing this for a single training example leads to an error of either zero or one, so it does not yet give an estimate of the test error. The leave-one-out method *repeats* this procedure for each individual training example in turn, and averages the resulting errors.

Let us return to the present case. If we leave out a pattern $\mathbf{x}_{i^*}$, and construct the solution from the remaining patterns, the following outcomes are possible (cf. (7.11)):

1. $y_{i^*}\left(\langle \mathbf{x}_{i^*}, \mathbf{w}\rangle + b\right) > 1$. In this case, the pattern is classified correctly and does not lie on the margin. These are patterns that would not have become SVs anyway.

2. $y_{i^*}\left(\langle \mathbf{x}_{i^*}, \mathbf{w}\rangle + b\right) = 1$. In other words, $\mathbf{x}_{i^*}$ exactly meets the constraint (7.11). In this case, the solution $\mathbf{w}$ does not change, even though the coefficients α_i would change: Namely, if $\mathbf{x}_{i^*}$ might have become a Support Vector (i.e., $\alpha_{i^*} > 0$) had it been kept in the training set. In that case, the fact that the solution is the same, no matter whether $\mathbf{x}_{i^*}$ is in the training set or not, means that $\mathbf{x}_{i^*}$ can be written as $\sum_{\mathrm{SVs}} \beta_i y_i \mathbf{x}_i$ with, $\beta_i \geq 0$. Note that condition 2 is *not* equivalent to saying that $\mathbf{x}_{i^*}$ may be written as some linear combination of the remaining Support Vectors: Since the sign of the coefficients in the linear combination is determined by the class of the respective pattern, not any linear combination will do. Strictly speaking, $\mathbf{x}_{i^*}$ must lie in the cone spanned by the $y_i \mathbf{x}_i$, where the $\mathbf{x}_i$ are all Support Vectors.[6] For more detail, see [565] and Section 12.2.

3. $0 < y_{i^*}\left(\langle \mathbf{x}_{i^*}, \mathbf{w}\rangle + b\right) < 1$. In this case, $\mathbf{x}_{i^*}$ lies within the margin, but still on the correct side of the decision boundary. Thus, the solution looks different from the one obtained with $\mathbf{x}_{i^*}$ in the training set (in that case, $\mathbf{x}_{i^*}$ would satisfy (7.11) after training); classification is nevertheless correct.

4. $y_{i^*}\left(\langle \mathbf{x}_{i^*}, \mathbf{w}\rangle + b\right) > 0$. This means that $\mathbf{x}_{i^*}$ is classified incorrectly.

Note that cases 3 and 4 necessarily correspond to examples which would have become SVs if kept in the training set; case 2 potentially includes such situations. Only case 4, however, leads to an error in the leave-one-out procedure. Consequently, we have the following result on the generalization error of optimal margin classifiers [570]:[7]

Leave-One-Out Bound

Proposition 7.4 *The expectation of the number of Support Vectors obtained during training on a training set of size m, divided by m, is an* upper bound *on the expected probability of test error of the SVM trained on training sets of size* $m - 1$.[8]

6. Possible non-uniqueness of the solution's expansion in terms of SVs is related to zero Eigenvalues of $(y_i y_j k(x_i, x_j))_{ij}$, cf. Proposition 2.16. Note, however, the above caveat on the distinction between linear combinations, and linear combinations with coefficients of fixed sign.

7. It also holds for the generalized versions of optimal margin classifiers described in the following sections.

8. Note that the leave-one-out procedure performed with m training examples thus yields

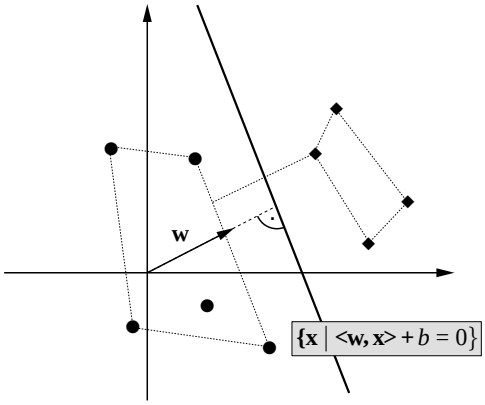


Figure 7.5 The optimal hyperplane (Figure 7.2) is the one bisecting the shortest connection between the convex hulls of the two classes.

A sharper bound can be formulated by making a further distinction in case 2, between SVs that must occur in the solution, and those that can be expressed in terms of the other SVs (see [570, 565, 268, 549] and Section 12.2).

We now return to the optimization problem to be solved. Substituting the conditions for the extremum, (7.14) and (7.15), into the Lagrangian (7.12), we arrive at the dual form of the optimization problem:

Quadratic Program of Optimal Margin Classifier

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^m}{\text{maximize}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \left\langle \mathbf{x}_i, \mathbf{x}_j \right\rangle, \tag{7.17}$$

$$\text{subject to} \quad \alpha_i \geq 0, \quad i = 1, \ldots, m, \tag{7.18}$$

$$\text{and} \quad \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{7.19}$$

On substitution of the expansion (7.15) into the decision function (7.3), we obtain an expression which can be evaluated in terms of dot products, taken between the pattern to be classified and the Support Vectors,

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i \left\langle \mathbf{x}, \mathbf{x}_i \right\rangle + b \right). \tag{7.20}$$

To conclude this section, we note that there is an alternative way to derive the dual optimization problem [38]. To describe it, we first form the convex hulls C_+

a bound valid for training sets of size $m-1$. This difference, however, does not usually mislead us too much. In statistical terms, the leave-one-out error is called *almost unbiased*. Note, moreover, that the statement talks about the *expected probability* of test error — there are thus *two* sources of randomness. One is the expectation over different training sets of size $m-1$, the other is the probability of test error when one of the SVMs is faced with a test example drawn from the underlying distribution generating the data. For a generalization, see Theorem 12.9.

and C_- of both classes of training points,

$$C_\pm := \left\{ \sum_{y_i=\pm 1} c_i \mathbf{x}_i \,\middle|\, \sum_{y_i=\pm 1} c_i = 1, c_i \geq 0 \right\}. \tag{7.21}$$

Convex Hull Separation

It can be shown that the maximum margin hyperplane as described above is the one bisecting the shortest line orthogonally connecting C_+ and C_- (Figure 7.5). Formally, this can be seen by considering the optimization problem

$$\begin{aligned} &\underset{\mathbf{c}\in\mathbb{R}^m}{\text{minimize}} \quad \left\| \sum_{y_i=1} c_i \mathbf{x}_i - \sum_{y_i=-1} c_i \mathbf{x}_i \right\|^2, \\ &\text{subject to} \quad \sum_{y_i=1} c_i = 1, \; \sum_{y_i=-1} c_i = 1, c_i \geq 0, \end{aligned} \tag{7.22}$$

and using the normal vector $\mathbf{w} = \sum_{y_i=1} c_i \mathbf{x}_i - \sum_{y_i=-1} c_i \mathbf{x}_i$, scaled to satisfy the canonicality condition (Definition 7.1). The threshold b is explicitly adjusted such that the hyperplane bisects the shortest connecting line (see also Problem 7.7).

7.4 Nonlinear Support Vector Classifiers

Thus far, we have shown why it is that a large margin hyperplane is good from a statistical point of view, and we have demonstrated how to compute it. Although these two points have worked out nicely, there is still a major drawback to the approach: Everything that we have done so far is linear in the data. To allow for much more general decision surfaces, we now use kernels to nonlinearly transform the input data $x_1, \ldots, x_m \in \mathcal{X}$ into a high-dimensional feature space, using a map $\Phi : x_i \mapsto \mathbf{x}_i$; we then do a linear separation there.

To justify this procedure, Cover's Theorem [113] is sometimes alluded to. This theorem characterizes the number of possible linear separations of m points in general position in an N-dimensional space. If $m \leq N+1$, then all 2^m separations are possible — the VC dimension of the function class is $n+1$ (Section 5.5.6). If $m > N+1$, then Cover's Theorem states that the number of linear separations equals

Cover's Theorem

$$2 \sum_{i=0}^{N} \binom{m-1}{i}. \tag{7.23}$$

The more we increase N, the more terms there are in the sum, and thus the larger is the resulting number. This theorem formalizes the intuition that the number of separations increases with the dimensionality. It requires, however, that the points are in general position — therefore, it does not strictly make a statement about the separability of a given dataset in a given feature space. E.g., the feature map might be such that all points lie on a rather restrictive lower-dimensional manifold, which could prevent us from finding points in general position.

There is another way to intuitively understand why the kernel mapping in-

and C_- of both classes of training points,

$$C_\pm := \left\{ \sum_{y_i=\pm 1} c_i \mathbf{x}_i \,\middle|\, \sum_{y_i=\pm 1} c_i = 1, c_i \geq 0 \right\}. \tag{7.21}$$

Convex Hull Separation

It can be shown that the maximum margin hyperplane as described above is the one bisecting the shortest line orthogonally connecting C_+ and C_- (Figure 7.5). Formally, this can be seen by considering the optimization problem

$$\begin{aligned} &\underset{\mathbf{c}\in\mathbb{R}^m}{\text{minimize}} \quad \left\| \sum_{y_i=1} c_i \mathbf{x}_i - \sum_{y_i=-1} c_i \mathbf{x}_i \right\|^2, \\ &\text{subject to} \quad \sum_{y_i=1} c_i = 1, \; \sum_{y_i=-1} c_i = 1, c_i \geq 0, \end{aligned} \tag{7.22}$$

and using the normal vector $\mathbf{w} = \sum_{y_i=1} c_i \mathbf{x}_i - \sum_{y_i=-1} c_i \mathbf{x}_i$, scaled to satisfy the canonicality condition (Definition 7.1). The threshold b is explicitly adjusted such that the hyperplane bisects the shortest connecting line (see also Problem 7.7).

7.4 Nonlinear Support Vector Classifiers

Thus far, we have shown why it is that a large margin hyperplane is good from a statistical point of view, and we have demonstrated how to compute it. Although these two points have worked out nicely, there is still a major drawback to the approach: Everything that we have done so far is linear in the data. To allow for much more general decision surfaces, we now use kernels to nonlinearly transform the input data $x_1, \ldots, x_m \in \mathcal{X}$ into a high-dimensional feature space, using a map $\Phi : x_i \mapsto \mathbf{x}_i$; we then do a linear separation there.

To justify this procedure, Cover's Theorem [113] is sometimes alluded to. This theorem characterizes the number of possible linear separations of m points in general position in an N-dimensional space. If $m \leq N+1$, then all 2^m separations are possible — the VC dimension of the function class is $n+1$ (Section 5.5.6). If $m > N+1$, then Cover's Theorem states that the number of linear separations equals

Cover's Theorem

$$2 \sum_{i=0}^{N} \binom{m-1}{i}. \tag{7.23}$$

The more we increase N, the more terms there are in the sum, and thus the larger is the resulting number. This theorem formalizes the intuition that the number of separations increases with the dimensionality. It requires, however, that the points are in general position — therefore, it does not strictly make a statement about the separability of a given dataset in a given feature space. E.g., the feature map might be such that all points lie on a rather restrictive lower-dimensional manifold, which could prevent us from finding points in general position.

There is another way to intuitively understand why the kernel mapping in-

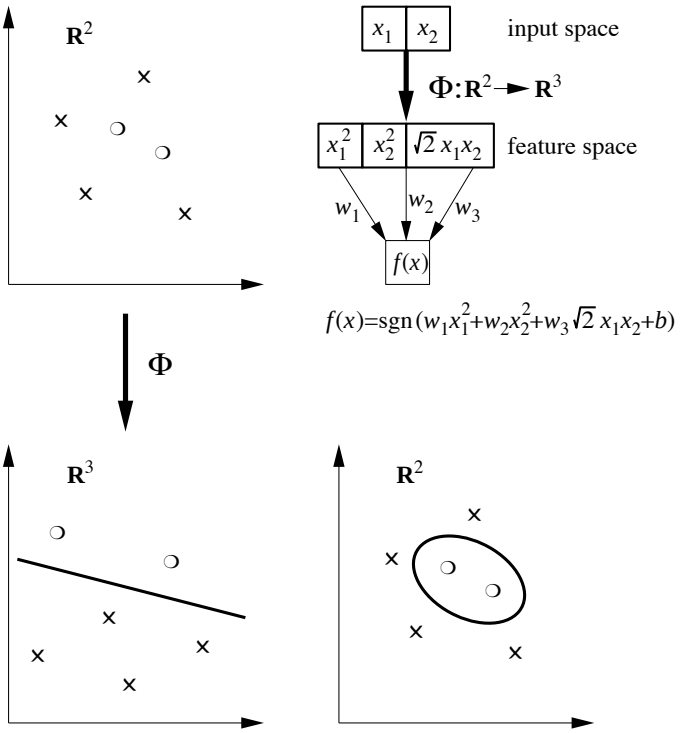


Figure 7.6 By mapping the input data *(top left)* nonlinearly (via Φ) into a higher-dimensional feature space $\mathcal{H}$ (here: $\mathcal{H} = \mathbb{R}^3$), and constructing a separating hyperplane there *(bottom left)*, an SVM *(top right)* corresponds to a nonlinear decision surface in input space (here: $\mathbb{R}^2$, *bottom right*). We use x_1, x_2 to denote the entries of the input vectors, and w_1, w_2, w_3 to denote the entries of the hyperplane normal vector in $\mathcal{H}$.

creases the chances of a separation, in terms of concepts of statistical learning theory. Using a kernel typically amounts to using a larger function class, thus increasing the capacity of the learning machine, and rendering problems separable that are not linearly separable to start with.

"Kernelizing" the Optimal Margin Hyperplane

On the practical level, the modification necessary to perform the algorithm in a high-dimensional feature space are minor. In the above sections, we made no assumptions on the dimensionality of $\mathcal{H}$, the space in which we assumed our patterns belong. We only required $\mathcal{H}$ to be equipped with a dot product. The patterns $\mathbf{x}_i$ that we talked about previously thus need not coincide with the input patterns. They can equally well be the results of mapping the original input patterns x_i into a high-dimensional feature space. Consequently, we take the stance that wherever we wrote $\mathbf{x}$, we actually meant $\Phi(x)$. Maximizing the target function (7.17), and evaluating the decision function (7.20), then requires the computation of dot products $\langle \Phi(x), \Phi(x_i) \rangle$ in a high-dimensional space. These expensive calculations are reduced significantly by using a positive definite kernel k (see Chapter 2), such that

Kernel Trick

$$\langle \Phi(x), \Phi(x_i) \rangle = k(x, x_i), \tag{7.24}$$

leading to decision functions of the form (cf. (7.20))

$$f(x) = \mathrm{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i k(x, x_i) + b \right). \tag{7.25}$$

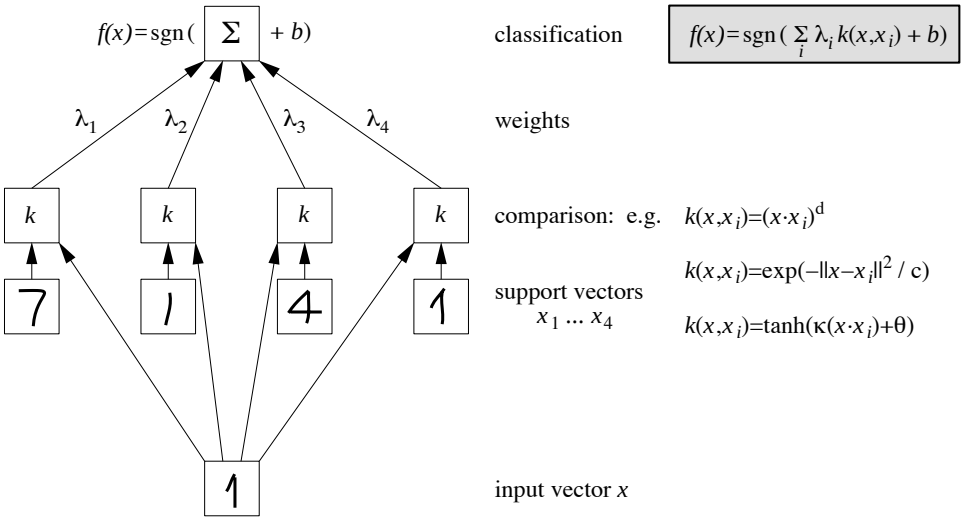


Figure 7.7 Architecture of SVMs. The kernel function k is chosen a priori; it determines the type of classifier (for instance, polynomial classifier, radial basis function classifier, or neural network). All other parameters (number of hidden units, weights, threshold b) are found during training, by solving a quadratic programming problem. The first layer weights x_i are a subset of the training set (the Support Vectors); the second layer weights $\lambda_i = y_i\alpha_i$ are computed from the Lagrange multipliers (cf. (7.25)).

At this point, a small aside regarding terminology is in order. As explained in Chapter 2, the input domain $\mathcal{X}$ need not be a vector space. Therefore, the Support Vectors in (7.25) (i.e., those x_i with $\alpha_i > 0$) are not necessarily vectors. One could choose to be on the safe side, and only refer to the corresponding $\Phi(x_i)$ as SVs. Common usage employs the term in a somewhat loose sense for both, however.

Consequently, everything that has been said about the linear case also applies to nonlinear cases, obtained using a suitable kernel k, instead of the Euclidean dot product (Figure 7.6). By using some of the kernel functions described in Chapter 2, the SV algorithm can construct a variety of learning machines (Figure 7.7), some of which coincide with classical architectures: *polynomial classifiers* of degree d,

Kernels

$$k(x, x_i) = \langle x, x_i \rangle^d, \tag{7.26}$$

radial basis function classifiers with Gaussian kernel of width $c > 0$,

$$k(x, x_i) = \exp\left(-\|x - x_i\|^2/c\right), \tag{7.27}$$

and *neural networks* (e.g., [49, 235]) with tanh activation function,

$$k(x, x_i) = \tanh(\kappa \langle x, x_i \rangle + \Theta). \tag{7.28}$$

The parameters $\kappa > 0$ and $\Theta \in \mathbb{R}$ are the gain and horizontal shift. As we shall see later, the tanh kernel can lead to very good results. Nevertheless, we should mention at this point that from a mathematical point of view, it has certain short-

comings, cf. the discussion following (2.69).

Quadratic Program

To find the decision function (7.25), we solve the following problem (cf. (7.17)):

$$\underset{\boldsymbol{\alpha}}{\text{maximize}}\, W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j), \tag{7.29}$$

subject to the constraints (7.18) and (7.19).

If k is positive definite, $Q_{ij} := (y_i y_j k(x_i, x_j))_{ij}$ is a positive definite matrix (Problem 7.6), which provides us with a convex problem that can be solved efficiently (cf. Chapter 6). To see this, note that (cf. Proposition 2.16)

$$\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) = \left\langle \sum_{i=1}^{m} \alpha_i y_i \Phi(x_i), \sum_{j=1}^{m} \alpha_j y_j \Phi(x_j) \right\rangle \geq 0, \tag{7.30}$$

for all $\boldsymbol{\alpha} \in \mathbb{R}^m$.

As described in Chapter 2, we can actually use a larger class of kernels without destroying the convexity of the quadratic program. This is due to the fact that the constraint (7.19) excludes certain parts of the space of multipliers α_i. As a result, we only need the kernel to be positive definite on the remaining points. This is precisely guaranteed if we require k to be *conditionally* positive definite (see Definition 2.21). In this case, we have $\boldsymbol{\alpha}^\top Q \boldsymbol{\alpha} \geq 0$ for all coefficient vectors $\boldsymbol{\alpha}$ satisfying (7.19).

Threshold

To compute the threshold b, we take into account that due to the KKT conditions (7.16), $\alpha_j > 0$ implies (using (7.24))

$$\sum_{i=1}^{m} y_i \alpha_i k(x_j, x_i) + b = y_j. \tag{7.31}$$

Thus, the threshold can for instance be obtained by averaging

$$b = y_j - \sum_{i=1}^{m} y_i \alpha_i k(x_j, x_i), \tag{7.32}$$

over all points with $\alpha_j > 0$; in other words, all SVs. Alternatively, one can compute b from the value of the corresponding double dual variable; see Section 10.3 for details. Sometimes it is also useful not to use the "optimal" b, but to change it in order to adjust the number of false positives and false negatives.

Figure 1.7 shows how a simple binary toy problem is solved, using a Support Vector Machine with a radial basis function kernel (7.27). Note that the SVs are the patterns closest to the decision boundary — not only in the feature space, where by construction, the SVs are the patterns closest to the separating hyperplane, but also in the input space depicted in the figure. This feature differentiates SVMs from other types of classifiers. Figure 7.8 shows both the SVs and the centers extracted by k-means, which are the expansion patterns that a classical RBF network approach would employ.

Comparison to RBF Network

In a study comparing the two approaches on the USPS problem of handwritten character recognition, a SVM with a Gaussian kernel outperformed the classical RBF network using Gaussian kernels [482]. A hybrid approach, where the SVM

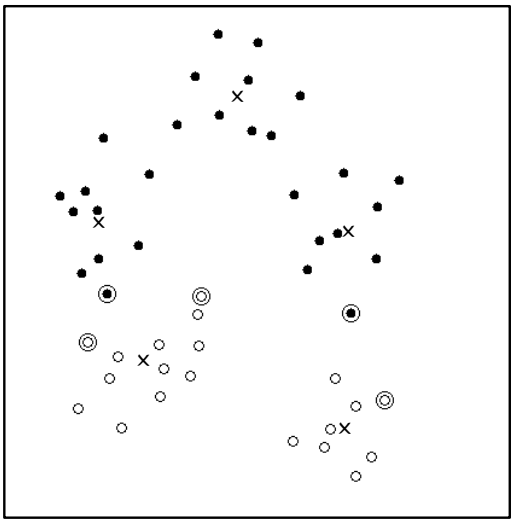


Figure 7.8 RBF centers automatically computed by the Support Vector algorithm (indicated by extra circles), using a Gaussian kernel. The number of SV centers accidentally coincides with the number of identifiable clusters (indicated by crosses found by k-means clustering, with $k = 2$ and $k = 3$ for balls and circles, respectively), but the naive correspondence between clusters and centers is lost; indeed, 3 of the SV centers are circles, and only 2 of them are balls. Note that the SV centers are chosen with respect to the classification task to be solved (from [482]).

algorithm was used to identify the centers (or hidden units) for the RBF network (that is, as a replacement for k-means), exhibited a performance which was in between the previous two. The study concluded that the SVM algorithm yielded two advantages. First, it better identified good expansion patterns, and second, its large margin regularizer led to second-layer weights that generalized better. We should add, however, that using clever engineering, the classical RBF algorithm can be improved to achieve a performance close to the one of SVMs [427].

7.5 Soft Margin Hyperplanes

So far, we have not said much about when the above will actually work. In practice, a separating hyperplane need not exist; and even if it does, it is not always the best solution to the classification problem. After all, an individual outlier in a data set, for instance a pattern which is mislabelled, can crucially affect the hyperplane. We would rather have an algorithm which can tolerate a certain fraction of outliers.

A natural idea might be to ask for the algorithm to return the hyperplane that leads to the *minimal* number of training errors. Unfortunately, it turns out that this is a combinatorial problem. Worse still, the problem is even hard to *approximate*: Ben-David and Simon [34] have recently shown that it is NP-hard to find a hyperplane whose training error is worse by some constant factor than the optimal one. Interestingly, they also show that this can be alleviated by taking into account the concept of the *margin*. By disregarding points that are within some fixed positive margin of the hyperplane, then the problem has polynomial complexity.

Cortes and Vapnik [111] chose a different approach for the SVM, following [40].

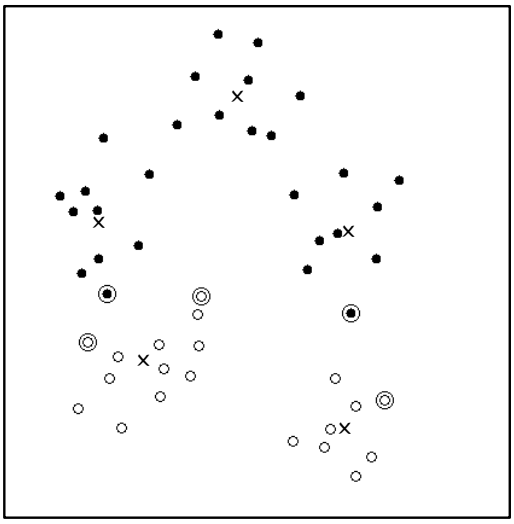


Figure 7.8 RBF centers automatically computed by the Support Vector algorithm (indicated by extra circles), using a Gaussian kernel. The number of SV centers accidentally coincides with the number of identifiable clusters (indicated by crosses found by k-means clustering, with $k = 2$ and $k = 3$ for balls and circles, respectively), but the naive correspondence between clusters and centers is lost; indeed, 3 of the SV centers are circles, and only 2 of them are balls. Note that the SV centers are chosen with respect to the classification task to be solved (from [482]).

algorithm was used to identify the centers (or hidden units) for the RBF network (that is, as a replacement for k-means), exhibited a performance which was in between the previous two. The study concluded that the SVM algorithm yielded two advantages. First, it better identified good expansion patterns, and second, its large margin regularizer led to second-layer weights that generalized better. We should add, however, that using clever engineering, the classical RBF algorithm can be improved to achieve a performance close to the one of SVMs [427].

7.5 Soft Margin Hyperplanes

So far, we have not said much about when the above will actually work. In practice, a separating hyperplane need not exist; and even if it does, it is not always the best solution to the classification problem. After all, an individual outlier in a data set, for instance a pattern which is mislabelled, can crucially affect the hyperplane. We would rather have an algorithm which can tolerate a certain fraction of outliers.

A natural idea might be to ask for the algorithm to return the hyperplane that leads to the *minimal* number of training errors. Unfortunately, it turns out that this is a combinatorial problem. Worse still, the problem is even hard to *approximate*: Ben-David and Simon [34] have recently shown that it is NP-hard to find a hyperplane whose training error is worse by some constant factor than the optimal one. Interestingly, they also show that this can be alleviated by taking into account the concept of the *margin*. By disregarding points that are within some fixed positive margin of the hyperplane, then the problem has polynomial complexity.

Cortes and Vapnik [111] chose a different approach for the SVM, following [40].

Slack Variables

To allow for the possibility of examples violating (7.11), they introduced so-called slack variables,

$$\xi_i \geq 0, \quad \text{where } i = 1, \ldots, m, \tag{7.33}$$

and use relaxed separation constraints (cf. (7.11)),

$$y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m. \tag{7.34}$$

Clearly, by making ξ_i large enough, the constraint on $(\mathbf{x}_i, y_i)$ can always be met. In order not to obtain the trivial solution where all ξ_i take on large values, we thus need to penalize them in the objective function. To this end, a term $\sum_i \xi_i$ is included in (7.10).

C-SVC

In the simplest case, referred to as the C-SV classifier, this is done by solving, for some $C > 0$,

$$\underset{\mathbf{w} \in \mathcal{H}, \boldsymbol{\xi} \in \mathbb{R}^m}{\text{minimize}} \quad \tau(\mathbf{w}, \boldsymbol{\xi}) = \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{m}\sum_{i=1}^{m} \xi_i, \tag{7.35}$$

subject to the constraints (7.33) and (7.34). It is instructive to compare this to Theorem 7.3, considering the case $\rho = 1$. Whenever the constraint (7.34) is met with $\xi_i = 0$, the corresponding point will not be a margin error. All non-zero slacks ξ correspond to margin errors; hence, roughly speaking, the fraction of margin errors in Theorem 7.3 increases with the second term in (7.35). The capacity term, on the other hand, increases with $\|\mathbf{w}\|$. Hence, for a suitable positive constant C, this approach approximately minimizes the right hand side of the bound.

Note, however, that if many of the ξ_i attain large values (in other words, if the classes to be separated strongly overlap, for instance due to noise), then $\sum_{i=1}^{m} \xi_i$ can be significantly larger than the fraction of margin errors. In that case, there is no guarantee that the hyperplane will generalize well.

As in the separable case (7.15), the solution can be shown to have an expansion

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i, \tag{7.36}$$

where non-zero coefficients α_i can only occur if the corresponding example $(\mathbf{x}_i, y_i)$ precisely meets the constraint (7.34). Again, the problem only depends on dot products in $\mathcal{H}$, which can be computed by means of the kernel.

The coefficients α_i are found by solving the following quadratic programming problem:

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximize}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j), \tag{7.37}$$

$$\text{subject to } 0 \leq \alpha_i \leq \frac{C}{m} \text{ for all } i = 1, \ldots, m, \tag{7.38}$$

$$\text{and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{7.39}$$

To compute the threshold b, we take into account that due to (7.34), for Support

Vectors x_j for which $\xi_j = 0$, we have (7.31). Thus, the threshold can be obtained by averaging (7.32) over all Support Vectors x_j (recall that they satisfy $\alpha_j > 0$) with $\alpha_j < C$.

In the above formulation, C is a constant determining the trade-off between two conflicting goals: minimizing the training error, and maximizing the margin. Unfortunately, C is a rather unintuitive parameter, and we have no a priori way to select it.[9] Therefore, a modification was proposed in [481], which replaces C by

ν-SVC

a parameter ν; the latter will turn out to control the number of margin errors and Support Vectors.

As a primal problem for this approach, termed the ν-SV classifier, we consider

$$\underset{\mathbf{w}\in\mathcal{H},\boldsymbol{\xi}\in\mathbb{R}^m,\rho,b\in\mathbb{R}}{\text{minimize}} \quad \tau(\mathbf{w},\boldsymbol{\xi},\rho) = \frac{1}{2}\|\mathbf{w}\|^2 - \nu\rho + \frac{1}{m}\sum_{i=1}^{m}\xi_i \tag{7.40}$$

$$\text{subject to} \quad y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) \geq \rho - \xi_i \tag{7.41}$$

$$\text{and} \quad \xi_i \geq 0, \quad \rho \geq 0. \tag{7.42}$$

Note that no constant C appears in this formulation; instead, there is a parameter ν, and also an additional variable ρ to be optimized. To understand the role of ρ, note that for $\boldsymbol{\xi} = 0$, the constraint (7.41) simply states that the two classes are separated by the *margin* $2\rho/\|\mathbf{w}\|$ (cf. Problem 7.4).

Margin Error

To explain the significance of ν, let us first recall the term *margin error*: by this, we denote points with $\xi_i > 0$. These are points which are either errors, or lie within the margin. Formally, the fraction of margin errors is

$$R^{\rho}_{\text{emp}}[g] := \frac{1}{m}\left|\{i|y_i g(x_i) < \rho\}\right|. \tag{7.43}$$

Here, g is used to denote the argument of the sgn in the decision function (7.25): $f = \text{sgn} \circ g$. We are now in a position to state a result that explains the significance

ν-Property

of ν.

Proposition 7.5 ([481]) *Suppose we run ν-SVC with k on some data with the result that $\rho > 0$. Then*

(i) ν is an upper bound on the fraction of margin errors.

(ii) ν is a lower bound on the fraction of SVs.

(iii) Suppose the data $(x_1, y_1), \ldots, (x_m, y_m)$ were generated iid from a distribution $\mathrm{P}(x, y) = \mathrm{P}(x)\mathrm{P}(y|x)$, such that neither $\mathrm{P}(x, y = 1)$ nor $\mathrm{P}(x, y = -1)$ contains any discrete component. Suppose, moreover, that the kernel used is analytic and non-constant. With probability 1, asymptotically, ν equals both the fraction of SVs and the fraction of errors.

The proof can be found in Section A.2.

Before we get into the technical details of the dual derivation, let us take a look

9. As a default value, we use $C/m = 10$ unless stated otherwise.

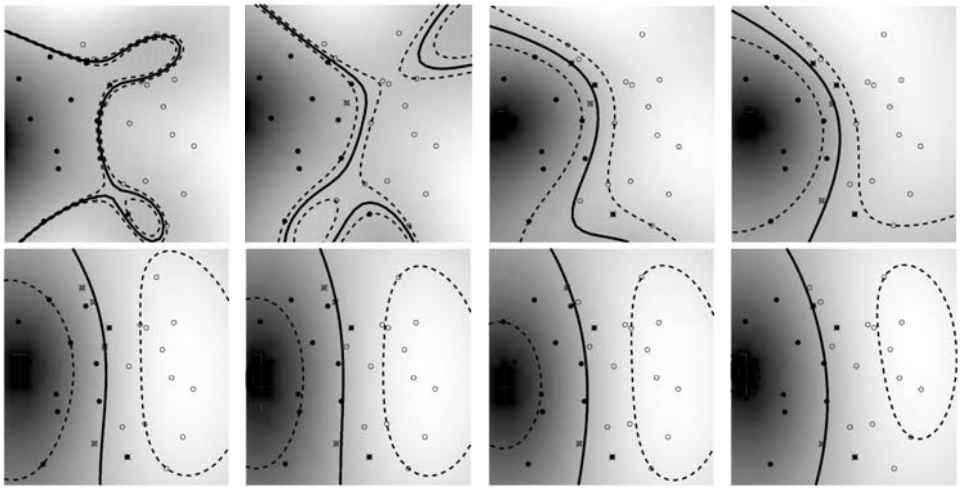


Figure 7.9 Toy problem (task: separate circles from disks) solved using ν-SV classification, with parameter values ranging from $\nu = 0.1$ (top left) to $\nu = 0.8$ (bottom right). The larger we make ν, the more points are allowed to lie inside the margin (depicted by dotted lines). Results are shown for a Gaussian kernel, $k(x, x') = \exp(-\|x - x'\|^2)$.

Table 7.1 Fractions of errors and SVs, along with the margins of class separation, for the toy example in Figure 7.9.
Note that ν upper bounds the fraction of errors and lower bounds the fraction of SVs, and that increasing ν, i.e., allowing more errors, increases the margin.

ν	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
fraction of errors	0.00	0.07	0.25	0.32	0.39	0.50	0.61	0.71
fraction of SVs	0.29	0.36	0.43	0.46	0.57	0.68	0.79	0.86
margin $\rho/\|\mathbf{w}\|$	0.005	0.018	0.115	0.156	0.364	0.419	0.461	0.546

at a toy example illustrating the influence of ν (Figure 7.9). The corresponding fractions of SVs and margin errors are listed in table 7.1.

Derivation of the Dual

The derivation of the ν-SVC dual is similar to the above SVC formulations, only slightly more complicated. We consider the Lagrangian

$$L(\mathbf{w}, \boldsymbol{\xi}, b, \rho, \boldsymbol{\alpha}, \boldsymbol{\beta}, \delta) = \frac{1}{2}\|\mathbf{w}\|^2 - \nu\rho + \frac{1}{m}\sum_{i=1}^{m}\xi_i \\ - \sum_{i=1}^{m}(\alpha_i(y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) - \rho + \xi_i) + \beta_i\xi_i) - \delta\rho, \tag{7.44}$$

using multipliers $\alpha_i, \beta_i, \delta \geq 0$. This function has to be minimized with respect to the primal variables $\mathbf{w}, \boldsymbol{\xi}, b, \rho$, and maximized with respect to the dual variables $\boldsymbol{\alpha}, \boldsymbol{\beta}, \delta$. To eliminate the former, we compute the corresponding partial derivatives

and set them to 0, obtaining the following conditions:

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i, \tag{7.45}$$

$$\alpha_i + \beta_i = 1/m, \tag{7.46}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0, \tag{7.47}$$

$$\sum_{i=1}^{m} \alpha_i - \delta = \nu. \tag{7.48}$$

Again, in the *SV expansion* (7.45), the α_i that are non-zero correspond to a constraint (7.41) which is precisely met.

Substituting (7.45) and (7.46) into L, using $\alpha_i, \beta_i, \delta \geq 0$, and incorporating kernels for dot products, leaves us with the following quadratic optimization problem for ν-SV classification:

Quadratic Program for ν-SVC

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximize}} \quad W(\boldsymbol{\alpha}) = -\frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j), \tag{7.49}$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq \frac{1}{m}, \tag{7.50}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0, \tag{7.51}$$

$$\sum_{i=1}^{m} \alpha_i \geq \nu. \tag{7.52}$$

As above, the resulting decision function can be shown to take the form

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i k(x, x_i) + b \right). \tag{7.53}$$

Compared with the C-SVC dual (7.37), there are two differences. First, there is an additional constraint (7.52).[10] Second, the linear term $\sum_{i=1}^{m} \alpha_i$ no longer appears in the objective function (7.49). This has an interesting consequence: (7.49) is now quadratically homogeneous in $\boldsymbol{\alpha}$. It is straightforward to verify that the same decision function is obtained if we start with the primal function

$$\tau(\mathbf{w}, \boldsymbol{\xi}, \rho) = \frac{1}{2}\|\mathbf{w}\|^2 + C\left(-\nu\rho + \frac{1}{m} \sum_{i=1}^{m} \xi_i \right), \tag{7.54}$$

10. The additional constraint makes it more challenging to come up with efficient training algorithms for large datasets. So far, two approaches have been proposed which work well. One of them slightly modifies the primal problem in order to avoid the *other* equality constraint (related to the offset b) [98]. The other one is a direct generalization of a corresponding algorithm for C-SVC, which reduces the problem for each chunk to a linear system, and which does not suffer any disadvantages from the additional constraint [407, 408]. See also Sections 10.3.2, 10.4.3, and 10.6.3 for further details.

i.e., if one does use C, cf. Problem 7.16.

To compute the threshold b and the margin parameter ρ, we consider two sets $S_\pm$, of identical size $s > 0$, containing SVs x_i with $0 < \alpha_i < 1$ and $y_i = \pm 1$, respectively. Then, due to the KKT conditions, (7.41) becomes an equality with $\xi_i = 0$. Hence, in terms of kernels,

$$b = -\frac{1}{2s} \sum_{x \in S_+ \cup S_-} \sum_{j=1}^{m} \alpha_j y_j k(x, x_j), \tag{7.55}$$

$$\rho = \frac{1}{2s} \Big(\sum_{x \in S_+} \sum_{j=1}^{m} \alpha_j y_j k(x, x_j) - \sum_{x \in S_-} \sum_{j=1}^{m} \alpha_j y_j k(x, x_j) \Big). \tag{7.56}$$

Note that for the decision function, only b is actually required.

Connection ν-SVC — C-SVC

A connection to standard SV classification, and a somewhat surprising interpretation of the regularization parameter C, is described by the following result:

Proposition 7.6 (Connection ν-SVC — C-SVC [481]) *If ν-SV classification leads to $\rho > 0$, then C-SV classification, with C set a priori to $1/\rho$, leads to the same decision function.*

Proof If we minimize (7.40), and then fix ρ to minimize only over the remaining variables, nothing will change. Hence the solution $\mathbf{w}_0, b_0, \boldsymbol{\xi}_0$ minimizes (7.35), for $C = 1$, subject to (7.41). To recover the constraint (7.34), we rescale to the set of variables $\mathbf{w}' = \mathbf{w}/\rho, b' = b/\rho, \boldsymbol{\xi}' = \boldsymbol{\xi}/\rho$. This leaves us with the objective function (7.35), up to a constant scaling factor ρ^2, using $C = 1/\rho$. ∎

For further details on the connection between ν-SVMs and C-SVMs, see [122, 38]. A complete account has been given by Chang and Lin [98], who show that for a given problem and kernel, there is an interval $[\nu_{\min}, \nu_{\max}]$ of admissible values for ν, with $0 \leq \nu_{\min} \leq \nu_{\max} \leq 1$. The boundaries of the interval are computed by considering $\sum_i \alpha_i$ as returned by the C-SVM in the limits $C \to \infty$ and $C \to 0$, respectively.

It has been noted that ν-SVMs have an interesting interpretation in terms of *reduced convex hulls* [122, 38] (cf. (7.21)). If a problem is non-separable, the convex hulls will no longer be disjoint. Therefore, it no longer makes sense to search for the shortest line connecting them, and the approach of (7.22) will fail. In this situation, it seems natural to reduce the convex hulls in size, by limiting the size of the coefficients c_i in (7.21) to some value $\nu \in (0, 1)$. Intuitively, this amounts to limiting the influence of individual points — note that in the original problem (7.22), two single points can already determine the solution. It is possible to show that the ν-SVM formulation solves the problem of finding the hyperplane orthogonal to the closest line connecting the *reduced* convex hulls [122].

Robustness and Outliers

We now move on to another aspect of soft margin classification. When we introduced the slack variables, we did not attempt to justify the fact that in the objective function, we used a penalizer $\sum_{i=1}^m \xi_i$. Why not use another penalizer, such as $\sum_{i=1}^m \xi_i^p$, for some $p \geq 0$ [111]? For instance, $p = 0$ would yield a penalizer

that exactly *counts* the number of margin errors. Unfortunately, however, it is also a penalizer that leads to a combinatorial optimization problem. Penalizers yielding optimization problems that are particularly convenient, on the other hand, are obtained for $p = 1$ and $p = 2$. By default, we use the former, as it possesses an additional property which is statistically attractive. As the following proposition shows, linearity of the target function in the slack variables ξ_i leads to a certain "outlier" resistance of the estimator. As above, we use the shorthand $\mathbf{x}_i$ for $\Phi(x_i)$.

Proposition 7.7 (Resistance of SV classification [481]) *Suppose* $\mathbf{w}$ *can be expressed in terms of the SVs which are not at bound,*

$$\mathbf{w} = \sum_{i=1}^{m} \gamma_i \mathbf{x}_i \tag{7.57}$$

with $\gamma_i \neq 0$ *only if* $\alpha_i \in (0, 1/m)$ *(where the* α_i *are the coefficients of the dual solution). Then local movements of any margin error* $\mathbf{x}_m$ *parallel to* $\mathbf{w}$ *do not change the hyperplane.*[11]

The proof can be found in Section A.2. For further results in support of the $p = 1$ case, see [527].

Note that the assumption (7.57) is not as restrictive as it may seem. Even though the SV expansion of the solution, $\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$, often contains many multipliers α_i which are at bound, it is nevertheless quite conceivable, especially when discarding the requirement that the coefficients be bounded, that we can obtain an expansion (7.57) in terms of a subset of the original vectors.

For instance, if we have a 2-D problem that we solve directly in input space, i.e., with $k(x, x') = \langle x, x' \rangle$, then it suffices to have two linearly independent SVs which are not at bound, in order to express $\mathbf{w}$. This holds true regardless of whether or not the two classes overlap, even if there are many SVs which are at the upper bound. Further information on resistance and robustness of SVMs can be found in Sections 3.4 and 9.3.

We have introduced SVs as those training examples x_i for which $\alpha_i > 0$. In some cases, it is useful to further distinguish different types of SVs. For reference purposes, we give a list of different types of SVs (Table 7.2).

In Section 7.3, we used the KKT conditions to argue that in the hard margin case, the SVs lie exactly on the margin. Using an identical argument for the soft margin case, we see that in this instance, in-bound SVs lie on the margin (Problem 7.9).

Note that in the hard margin case, where $\alpha_{\max} = \infty$, every SV is an in-bound SV. Note, moreover, that for kernels that produce full-rank Gram matrices, such as the Gaussian (Theorem 2.18), in theory every SV is essential (provided there are no duplicate patterns in the training set).[12]

11. Note that the perturbation of the point is carried out in feature space. What it precisely corresponds to in input space therefore depends on the specific kernel chosen.
12. In practice, Gaussian Gram matrices usually have some eigenvalues that are close to 0.

Table 7.2 Overview of different types of SVs. In each case, the condition on the Lagrange multipliers α_i (corresponding to an SV x_i) is given. In the table, $\alpha_{\max}$ stands for the upper bound in the optimization problem; for instance, $\alpha_{\max} = \frac{C}{m}$ in (7.38) and $\alpha_{\max} = \frac{1}{m}$ in (7.50).

Type of SV	Definition	Properties
(standard) SV	$0 < \alpha_i$	lies on or in margin
in-bound SV	$0 < \alpha_i < \alpha_{\max}$	lies on margin
bound SV	$\alpha_i = \alpha_{\max}$	usually lies in margin ("margin error")
essential SV	appears in all possible expansions of solution	becomes margin error when left out (Section 7.3)

7.6 Multi-Class Classification

So far, we have talked about binary classification, where the class labels can only take two values: ± 1. Many real-world problems, however, have more than two classes — an example being the widely studied optical character recognition (OCR) problem. We will now review some methods for dealing with this issue.

7.6.1 One Versus the Rest

To get *M-class classifiers*, it is common to construct a set of binary classifiers $f^1, \ldots, f^M$, each trained to separate one class from the rest, and combine them by doing the multi-class classification according to the maximal output before applying the sgn function; that is, by taking

$$\operatorname*{argmax}_{j=1,\ldots,M} g^j(x), \quad \text{where } g^j(x) = \sum_{i=1}^{m} y_i \alpha_i^j k(x, x_i) + b^j \tag{7.58}$$

(note that $f^j(x) = \operatorname{sgn}(g^j(x))$, cf. (7.25)).

Reject Decisions

The values $g^j(x)$ can also be used for *reject decisions*. To see this, we consider the difference between the two largest $g^j(x)$ as a measure of confidence in the classification of x. If that measure falls short of a threshold θ, the classifier rejects the pattern and does not assign it to a class (it might instead be passed on to a human expert). This has the consequence that on the remaining patterns, a lower error rate can be achieved. Some benchmark comparisons report a quantity referred to as the *punt error*, which denotes the fraction of test patterns that must be rejected in order to achieve a certain accuracy (say 1% error) on the remaining test samples. To compute it, the value of θ is adjusted on the *test* set [64].

The main shortcoming of (7.58), sometimes called the *winner-takes-all* approach, is that it is somewhat heuristic. The binary classifiers used are obtained by training on different binary classification problems, and thus it is unclear whether their

Table 7.2 Overview of different types of SVs. In each case, the condition on the Lagrange multipliers α_i (corresponding to an SV x_i) is given. In the table, $\alpha_{\max}$ stands for the upper bound in the optimization problem; for instance, $\alpha_{\max} = \frac{C}{m}$ in (7.38) and $\alpha_{\max} = \frac{1}{m}$ in (7.50).

Type of SV	Definition	Properties
(standard) SV	$0 < \alpha_i$	lies on or in margin
in-bound SV	$0 < \alpha_i < \alpha_{\max}$	lies on margin
bound SV	$\alpha_i = \alpha_{\max}$	usually lies in margin ("margin error")
essential SV	appears in all possible expansions of solution	becomes margin error when left out (Section 7.3)

7.6 Multi-Class Classification

So far, we have talked about binary classification, where the class labels can only take two values: ± 1. Many real-world problems, however, have more than two classes — an example being the widely studied optical character recognition (OCR) problem. We will now review some methods for dealing with this issue.

7.6.1 One Versus the Rest

To get *M-class classifiers*, it is common to construct a set of binary classifiers $f^1, \ldots, f^M$, each trained to separate one class from the rest, and combine them by doing the multi-class classification according to the maximal output before applying the sgn function; that is, by taking

$$\operatorname*{argmax}_{j=1,\ldots,M} g^j(x), \quad \text{where } g^j(x) = \sum_{i=1}^{m} y_i \alpha_i^j k(x, x_i) + b^j \tag{7.58}$$

(note that $f^j(x) = \operatorname{sgn}(g^j(x))$, cf. (7.25)).

Reject Decisions

The values $g^j(x)$ can also be used for *reject decisions*. To see this, we consider the difference between the two largest $g^j(x)$ as a measure of confidence in the classification of x. If that measure falls short of a threshold θ, the classifier rejects the pattern and does not assign it to a class (it might instead be passed on to a human expert). This has the consequence that on the remaining patterns, a lower error rate can be achieved. Some benchmark comparisons report a quantity referred to as the *punt error*, which denotes the fraction of test patterns that must be rejected in order to achieve a certain accuracy (say 1% error) on the remaining test samples. To compute it, the value of θ is adjusted on the *test* set [64].

The main shortcoming of (7.58), sometimes called the *winner-takes-all* approach, is that it is somewhat heuristic. The binary classifiers used are obtained by training on different binary classification problems, and thus it is unclear whether their

real-valued outputs (before thresholding) are on comparable scales.[13] This can be a problem, since situations often arise where *several* binary classifiers assign the pattern to their respective class (or where *none* does); in this case, *one* class must be chosen by comparing the real-valued outputs.

In addition, binary one-versus-the-rest classifiers have been criticized for dealing with rather asymmetric problems. For instance, in digit recognition, the classifier trained to recognize class '7' is usually trained on many more negative than positive examples. We can deal with these asymmetries by using values of the regularization constant C which differ for the respective classes (see Problem 7.10). It has nonetheless been argued that the following approach, which is more symmetric from the outset, can be advantageous.

7.6.2 Pairwise Classification

In pairwise classification, we train a classifier for each possible pair of classes [178, 463, 233, 311]. For M classes, this results in $(M-1)M/2$ binary classifiers. This number is usually larger than the number of one-versus-the-rest classifiers; for instance, if $M = 10$, we need to train 45 binary classifiers rather than 10 as in the method above. Although this suggests large training times, the individual problems that we need to train on are significantly smaller, and if the training algorithm scales superlinearly with the training set size, it is actually possible to save time.

Similar considerations apply to the runtime execution speed. When we try to classify a test pattern, we evaluate all 45 binary classifiers, and classify according to which of the classes gets the highest number of votes. A vote for a given class is defined as a classifier putting the pattern into that class.[14] The individual classifiers, however, are usually smaller in size (they have fewer SVs) than they would be in the one-versus-the-rest approach. This is for two reasons: First, the training sets are smaller, and second, the problems to be learned are usually easier, since the classes have less overlap.

Nevertheless, if M is large, and we evaluate the $(M-1)M/2$ classifiers, then the resulting system may be slower than the corresponding one-versus-the-rest SVM. To illustrate this weakness, consider the following hypothetical situation: Suppose, in a digit recognition task, that after evaluating the first few binary classifiers, both digit 7 and digit 8 seem extremely unlikely (they already "lost" on several classifiers). In that case, it would seem pointless to evaluate the 7-vs-8 classifier. This idea can be cast into a precise framework by embedding the binary classifiers into a directed acyclic graph. Each classification run then corresponds to a directed traversal of that graph, and classification can be much faster [411].

---

13. Note, however, that some effort has gone into developing methods for transforming the real-valued outputs into class probabilities [521, 486, 410].
14. Some care has to be exercised in tie-breaking. For further detail, see [311].

7.6.3 Error-Correcting Output Coding

The method of error-correcting output codes was developed in [142], and later adapted to the case of SVMs [5]. In a nutshell, the idea is as follows. Just as we can generate a binary problem from a multiclass problem by separating one class from the rest — digit 0 from digits 1 through 9, say — we can generate a large number of further binary problems by splitting the original set of classes into two subsets. For instance, we could separate the even digits from the odd ones, or we could separate digits 0 through 4 from 5 through 9. It is clear that if we design a set of binary classifiers $f^1, \ldots, f^L$ in the right way, then the binary responses will completely determine the class of a test patterns. Each class corresponds to a unique vector in $\{\pm 1\}^L$; for M classes, we thus get a so-called *decoding matrix* $M \in \{\pm 1\}^{M \times L}$. What happens if the binary responses are inconsistent with each other; if, for instance, the problem is noisy, or the training sample is too small to estimate the binary classifiers reliably? Formally, this means that we will obtain a vector of responses $f^1(x), \ldots, f^L(x)$ which does not occur in the matrix M. To deal with these cases, [142] proposed designing a clever set of binary problems, which yields robustness against some errors. Here, the closest match between the vector of responses and the rows of the matrix is determined using the Hamming distance (the number of entries where the two vectors differ; essentially, the L_∞ distance). Now imagine a situation where the code is such that the minimal Hamming distance is three. In this case, we can *guarantee* that we will correctly classify all test examples which lead to at most one error amongst the binary classifiers.

This method produces very good results in multi-class tasks; nevertheless, it has been pointed out that it does not make use of a crucial quantity in classifiers: the margin. Recently [5], a version was developed that replaces the Hamming-based decoding with a more sophisticated scheme that takes margins into account. Recommendations are also made regarding how to design good codes for margin classifiers, such as SVMs.

7.6.4 Multi-Class Objective Functions

Arguably the most elegant multi-class algorithm, and certainly the method most closely aligned with Vapnik's principle of always trying to solve problems *directly*, entails modifying the SVM objective function in such a way that it simultaneously allows the computation of a multi-class classifier. For instance [593, 58], we can modify (7.35) and use the following quadratic program:

$$\underset{\mathbf{w}_r \in \mathcal{H}, \boldsymbol{\xi}^r \in \mathbb{R}^m, b_r \in \mathbb{R}}{\text{minimize}} \ \frac{1}{2} \sum_{r=1}^{M} \|\mathbf{w}_r\|^2 + \frac{C}{m} \sum_{i=1}^{m} \sum_{r \neq y_i} \xi_i^r, \tag{7.59}$$

$$\text{subject to } \langle \mathbf{w}_{y_i}, \mathbf{x}_i \rangle + b_{y_i} \geq \langle \mathbf{w}_r, \mathbf{x}_i \rangle + b_r + 2 - \xi_i^r,$$

$$\xi_i^r \geq 0, \tag{7.60}$$

where $m \in \{1, \ldots, M\} \setminus y_i$, and $y_i \in \{1, \ldots, M\}$ is the multi-class label of the pattern $\mathbf{x}_i$ (cf. Problem 7.17).

In terms of accuracy, the results obtained with this approach are comparable to those obtained with the widely used one-versus-the-rest approach. Unfortunately, the optimization problem is such that it has to deal with *all* SVs at the same time. In the other approaches, the individual binary classifiers usually have much smaller SV sets, with beneficial effects on the training time. For further multiclass approaches, see [160, 323]. Generalizations to *multi-label* problems, where patterns are allowed to belong to several classes at the same time, are discussed in [162].

Overall, it is fair to say that there is probably no multi-class approach that generally outperforms the others. For practical problems, the choice of approach will depend on constraints at hand. Relevant factors include the required accuracy, the time available for development and training, and the nature of the classification problem (e.g., for a problem with very many classes, it would not be wise to use (7.59)). That said, a simple one-against-the-rest approach often produces acceptable results.

7.7 Variations on a Theme

There are a number of variations of the standard SV classification algorithm, such as the elegant *leave-one-out machine* [589, 592] (see also Section 12.2.2 below), the idea of *Bayes point machines* [451, 239, 453, 545, 392], and extensions to *feature selection* [70, 224, 590]. Due to lack of space, we only describe one of the variations; namely, *linear programming machines*.

Linear Programming Machines

As we have seen above, the SVM approach automatically leads to a decision function of the form (7.25). Let us rewrite it as $f(x) = \operatorname{sgn}(g(x))$, with

$$g(x) = \sum_{i=1}^{m} v_i k(x, x_i) + b. \tag{7.61}$$

In Chapter 4, we showed that this form of the solution is essentially a consequence of the form of the regularizer $\|\mathbf{w}\|^2$ (Theorem 4.2). The idea of linear programming (LP) machines is to use the kernel expansion as an ansatz for the solution, but to use a different regularizer, namely the ℓ_1 norm of the coefficient vector [343, 344, 74, 184, 352, 37, 591, 593, 39]. The main motivation for this is that this regularizer is known to induce sparse expansions (see Chapter 4).

ℓ_1 Regularizer

This amounts to the objective function

$$R_{\text{reg}}[g] := \frac{1}{m}\|\boldsymbol{v}\|_1 + C\, R_{\text{emp}}[g], \tag{7.62}$$

where $\|\boldsymbol{v}\|_1 = \sum_{i=1}^{m} |v_i|$ denotes the ℓ_1 norm in coefficient space, using the soft margin empirical risk,

$$R_{\text{emp}}[g] = \frac{1}{m} \sum_i \xi_i, \tag{7.63}$$

where $m \in \{1, \ldots, M\} \setminus y_i$, and $y_i \in \{1, \ldots, M\}$ is the multi-class label of the pattern $\mathbf{x}_i$ (cf. Problem 7.17).

In terms of accuracy, the results obtained with this approach are comparable to those obtained with the widely used one-versus-the-rest approach. Unfortunately, the optimization problem is such that it has to deal with *all* SVs at the same time. In the other approaches, the individual binary classifiers usually have much smaller SV sets, with beneficial effects on the training time. For further multiclass approaches, see [160, 323]. Generalizations to *multi-label* problems, where patterns are allowed to belong to several classes at the same time, are discussed in [162].

Overall, it is fair to say that there is probably no multi-class approach that generally outperforms the others. For practical problems, the choice of approach will depend on constraints at hand. Relevant factors include the required accuracy, the time available for development and training, and the nature of the classification problem (e.g., for a problem with very many classes, it would not be wise to use (7.59)). That said, a simple one-against-the-rest approach often produces acceptable results.

7.7 Variations on a Theme

There are a number of variations of the standard SV classification algorithm, such as the elegant *leave-one-out machine* [589, 592] (see also Section 12.2.2 below), the idea of *Bayes point machines* [451, 239, 453, 545, 392], and extensions to *feature selection* [70, 224, 590]. Due to lack of space, we only describe one of the variations; namely, *linear programming machines*.

Linear Programming Machines

As we have seen above, the SVM approach automatically leads to a decision function of the form (7.25). Let us rewrite it as $f(x) = \operatorname{sgn}(g(x))$, with

$$g(x) = \sum_{i=1}^{m} v_i k(x, x_i) + b. \tag{7.61}$$

In Chapter 4, we showed that this form of the solution is essentially a consequence of the form of the regularizer $\|\mathbf{w}\|^2$ (Theorem 4.2). The idea of linear programming (LP) machines is to use the kernel expansion as an ansatz for the solution, but to use a different regularizer, namely the ℓ_1 norm of the coefficient vector [343, 344, 74, 184, 352, 37, 591, 593, 39]. The main motivation for this is that this regularizer is known to induce sparse expansions (see Chapter 4).

ℓ_1 Regularizer

This amounts to the objective function

$$R_{\text{reg}}[g] := \frac{1}{m}\|\boldsymbol{v}\|_1 + C\, R_{\text{emp}}[g], \tag{7.62}$$

where $\|\boldsymbol{v}\|_1 = \sum_{i=1}^{m} |v_i|$ denotes the ℓ_1 norm in coefficient space, using the soft margin empirical risk,

$$R_{\text{emp}}[g] = \frac{1}{m} \sum_i \xi_i, \tag{7.63}$$

with slack terms

$$\xi_i = \max\{1 - y_i g(x_i), 0\}. \tag{7.64}$$

We thus obtain a linear programming problem;

$$\begin{array}{ll} \underset{\boldsymbol{\alpha},\boldsymbol{\xi}\in\mathbb{R}^m,b\in\mathbb{R}}{\text{minimize}} & \frac{1}{m}\sum_{i=1}^{m}(\alpha_i+\alpha_i^*)+C\sum_{i=1}^{m}\xi_i, \\ \text{subject to} & y_i g(x_i) \geq 1-\xi_i, \\ & \alpha_i, \alpha_i^*, \xi_i \geq 0. \end{array} \tag{7.65}$$

Here, we have dealt with the ℓ_1-norm by splitting each component v_i into its positive and negative part: $v_i = \alpha_i - \alpha_i^*$ in (7.61). The solution differs from (7.25) in that it is no longer necessarily the case that each expansion pattern has a weight $\alpha_i y_i$, whose sign equals its class label. This property would have to be enforced separately (Problem 7.19). Moreover, it is also no longer the case that the expansion patterns lie on or beyond the margin — in LP machines, they can basically be anywhere.

ν-LPMs

LP machines can also benefit from the ν-trick. In this case, the programming problem can be shown to take the following form [212]:

$$\begin{array}{ll} \underset{\boldsymbol{\alpha},\boldsymbol{\xi}\in\mathbb{R}^m,b,\rho\in\mathbb{R}}{\text{minimize}} & \frac{1}{m}\sum_{i=1}^{m}\xi_i - \nu\rho, \\ \text{subject to} & \frac{1}{m}\sum_{i=1}^{m}(\alpha_i+\alpha_i^*)=1, \\ & y_i g(\mathbf{x}_i) \geq \rho - \xi_i, \\ & \alpha_i, \alpha_i^*, \xi_i, \rho \geq 0. \end{array} \tag{7.66}$$

We will not go into further detail at this point. Additional information on linear programming machines from a regularization point of view is given in Section 4.9.2.

7.8 Experiments

7.8.1 Digit Recognition Using Different Kernels

Handwritten digit recognition has long served as a test bed for evaluating and benchmarking classifiers [318, 64, 319]. Thus, it was imperative in the early days of SVM research to evaluate the SV method on widely used digit recognition tasks. In this section we report results on the US Postal Service (USPS) database (described in Section A.1). We shall return to the character recognition problem in Chapter 11, where we consider the larger MNIST database.

As described above, the difference between C-SVC and ν-SVC lies only in the fact that we have to select a different parameter a priori. If we are able to do this

with slack terms

$$\xi_i = \max\{1 - y_i g(x_i), 0\}. \tag{7.64}$$

We thus obtain a linear programming problem;

$$\begin{array}{ll} \underset{\boldsymbol{\alpha},\boldsymbol{\xi}\in\mathbb{R}^m,b\in\mathbb{R}}{\text{minimize}} & \frac{1}{m}\sum_{i=1}^{m}(\alpha_i+\alpha_i^*)+C\sum_{i=1}^{m}\xi_i, \\ \text{subject to} & y_i g(x_i) \geq 1-\xi_i, \\ & \alpha_i, \alpha_i^*, \xi_i \geq 0. \end{array} \tag{7.65}$$

Here, we have dealt with the ℓ_1-norm by splitting each component v_i into its positive and negative part: $v_i = \alpha_i - \alpha_i^*$ in (7.61). The solution differs from (7.25) in that it is no longer necessarily the case that each expansion pattern has a weight $\alpha_i y_i$, whose sign equals its class label. This property would have to be enforced separately (Problem 7.19). Moreover, it is also no longer the case that the expansion patterns lie on or beyond the margin — in LP machines, they can basically be anywhere.

ν-LPMs

LP machines can also benefit from the ν-trick. In this case, the programming problem can be shown to take the following form [212]:

$$\begin{array}{ll} \underset{\boldsymbol{\alpha},\boldsymbol{\xi}\in\mathbb{R}^m,b,\rho\in\mathbb{R}}{\text{minimize}} & \frac{1}{m}\sum_{i=1}^{m}\xi_i - \nu\rho, \\ \text{subject to} & \frac{1}{m}\sum_{i=1}^{m}(\alpha_i+\alpha_i^*)=1, \\ & y_i g(\mathbf{x}_i) \geq \rho - \xi_i, \\ & \alpha_i, \alpha_i^*, \xi_i, \rho \geq 0. \end{array} \tag{7.66}$$

We will not go into further detail at this point. Additional information on linear programming machines from a regularization point of view is given in Section 4.9.2.

7.8 Experiments

7.8.1 Digit Recognition Using Different Kernels

Handwritten digit recognition has long served as a test bed for evaluating and benchmarking classifiers [318, 64, 319]. Thus, it was imperative in the early days of SVM research to evaluate the SV method on widely used digit recognition tasks. In this section we report results on the US Postal Service (USPS) database (described in Section A.1). We shall return to the character recognition problem in Chapter 11, where we consider the larger MNIST database.

As described above, the difference between C-SVC and ν-SVC lies only in the fact that we have to select a different parameter a priori. If we are able to do this

Table 7.3 Performance on the USPS set, for three different types of classifier, constructed with the Support Vector algorithm by choosing different functions k in (7.25) and (7.29). Error rates on the test set are given; and for each of the ten-class-classifiers, we also show the average number of Support Vectors of the ten two-class-classifiers. The normalization factor of 256 is tailored to the dimensionality of the data, which is 16×16.

polynomial: $k(x, x') = \left(\langle x, x' \rangle / 256\right)^d$

d	1	2	3	4	5	6	7
raw error/%	8.9	4.7	4.0	4.2	4.5	4.5	4.7
av. # of SVs	282	237	274	321	374	422	491

RBF: $k(x, x') = \exp\left(-\|x - x'\|^2/(256\ c)\right)$

c	4.0	2.0	1.2	0.8	0.5	0.2	0.1
raw error/%	5.3	5.0	4.9	4.3	4.4	4.4	4.5
av. # of SVs	266	240	233	235	251	366	722

sigmoid: $k(x, x') = \tanh(2\langle x, x' \rangle / 256 + \Theta)$

$-\Theta$	0.8	0.9	1.0	1.1	1.2	1.3	1.4
raw error/%	6.3	4.8	4.1	4.3	4.3	4.4	4.8
av. # of SVs	206	242	254	267	278	289	296

well, we obtain identical performance. The experiments reported were carried out before the development of ν-SVC, and thus all use C-SVC code.

In the present study, we put particular emphasis on comparing different types of SV classifiers obtained by choosing different kernels. We report results for polynomial kernels (7.26), Gaussian radial basis function kernels (7.27), and sigmoid kernels (7.28), summarized in Table 7.3. In all three cases, error rates around 4% can be achieved.

Kernel Scaling

Note that in practical applications, it is usually helpful to scale the argument of the kernel, such that the numerical values do not get extremely small or large as the dimension of the data increases. This helps avoid large roundoff errors, and prevents over- and underflow. In the present case, the scaling was done by including the factor 256 in Table 7.3.

The results show that the Support Vector algorithm allows the construction of a range of learning machines, all of which perform well. The similar performance for the three different functions k suggests that among these cases, the choice of the set of decision functions is less important than capacity control in the chosen type of structure. This phenomenon is well-known for the Parzen window density estimator in $\mathbb{R}^N$ (e.g., [226])

$$p(x) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{\omega^N} k\left(\frac{x - x_i}{\omega}\right). \tag{7.67}$$

It is of great importance in this case to choose an appropriate value of the band-

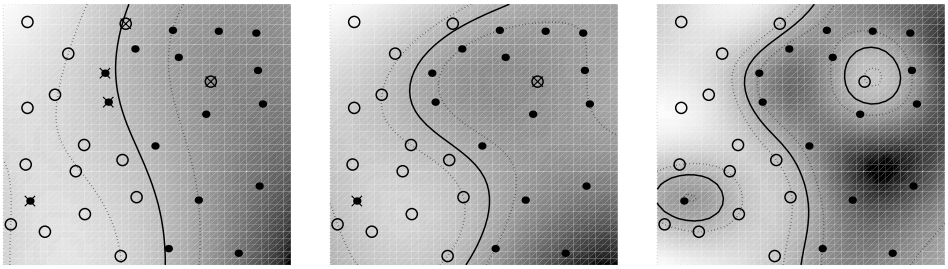


Figure 7.10 2D toy example of a binary classification problem solved using a soft margin SVC. In all cases, a Gaussian kernel (7.27) is used. From left to right, we decrease the kernel width. Note that for a large width, the decision boundary is almost linear, and the data set cannot be separated without error (see text). Solid lines represent decision boundaries; dotted lines depict the edge of the margin (where (7.34) becomes an equality with $\xi_i = 0$).

width parameter ω for a given amount of data. Similar parallels can be drawn to the solution of ill-posed problems; for a discussion, see [561].

Figure 7.10 shows a toy example using a Gaussian kernel (7.27), illustrating that it is crucial to pick the right kernel parameter. In all cases, the same value of C was used, but the kernel width c was varied. For large values of c, the classifier is almost linear, and it cannot separate the data set without errors. For a small width (right), the data set is practically *memorized*. For an intermediate width (middle), a trade-off is made between allowing *some* training errors and using a "simple" decision boundary.

Parameter Choice

In practice, both the kernel parameters and the value of C (or ν) are often chosen using *cross validation*. To this end, we first split the data set into p parts of equal size, say, $p = 10$. We then perform ten training runs. Each time, we leave out one of the ten parts and use it as an independent validation set for optimizing the parameters. In the simplest case, we choose the parameters which work best, on average over the ten runs. It is common practice, however, to then train on the full training set, using these average parameters. There are some problems with this. First, it amounts to optimizing the parameters on the same set as the one used for training, which can lead to overfitting. Second, the optimal parameter settings for data sets of size m and $\frac{9}{10}m$, respectively, do not usually coincide. Typically, the smaller set will require a slightly stronger regularization. This could mean a wider Gaussian kernel, a smaller polynomial degree, a smaller C, or a larger ν. Even worse, it is theoretically possible that there is a so-called phase transition (e.g., [393]) in the learning curve between the two sample sizes. This means that the generalization error as a function of the sample size could change dramatically between $\frac{9}{10}m$ and m. Having said all this, practitioners often do not care about these theoretical precautions, and use the unchanged parameters with excellent results. For further detail, see Section 12.2.

In some cases, one can try to avoid the whole procedure by using an educated guess. Below, we list several methods.

- Use parameter settings that have worked well for similar problems. Here, some care has to be exercised in the scaling of kernel parameters. For instance, when using an RBF kernel, c must be rescaled to ensure that $\|x_i - x_j\|^2/c$ roughly lies in the same range, even if the scaling and dimension of the data are different.
- For many problems, there is some prior expectation regarding the typical error rate. Let us assume we are looking at an image classification task, and we have already tried three other approaches, all of which yielded around 5% test error. Using ν–SV classifiers, we can incorporate this knowledge by choosing a value for ν which is in that range, say $\nu = 5\%$. The reason for this guess is that we know (Proposition 7.5) that the margin error is then below 5%, which in turn implies that the training error is below 5%. The training error will typically be smaller than the test error, thus it is consistent that it should be upper bounded by the 5% test error.
- In a slightly less elegant way, one can try to mimic this procedure for C-SV classifiers. To this end, we start off with a large value of C, and reduce it until the number of Lagrange multipliers that are at the upper bound (in other words, the number of margin errors) is in a suitable range (say, somewhat below 5%). Compared to the above procedure for choosing ν, the disadvantage is that this entails a number of training runs. We can also monitor the number of actual training errors during the training runs, but since not every margin error is a training error, this is often less sensitive. Indeed, the difference between training error and test error can often be quite substantial. For instance, on the USPS set, most of the results reported here were obtained with systems that had essentially zero training error.
- One can put forward scaling arguments which indicate that $C \propto 1/R^2$, where R is a measure for the range of the data in feature space that scales like the length of the points in $\mathcal{H}$. Examples thereof are the standard deviation of the distance of the points to their mean, the radius of the smallest sphere containing the data (cf. (5.61) and (8.17)), or, in some cases, the maximum (or mean) length $k(x_i, x_i)$ over all data points (see Problem 7.25).
- Finally, we can use theoretical tools such as VC bounds (see, for instance, Figure 5.5) or leave-one-out bounds (Section 12.2).

Having seen that different types of SVCs lead to similar performance, the question arises as to how these performances compare with other approaches. Table 7.4 gives a summary of a number of results on the USPS set. Note that the best SVM result is 3.0%; it uses additional techniques that we shall explain in chapters 11 and 13. It is known that the USPS test set is rather difficult — the human error rate is 2.5% [79]. For a discussion, see [496]. Note, moreover, that some of the results reported in the literature for the USPS set were obtained with an enhanced training set: For instance, the study of Drucker et al. [148] used an enlarged training set of size 9709, containing some additional machine-printed digits, and found that this improves the accuracy on the test set. Similarly, Bottou and Vapnik [65] used a training set of size 9840. Since there are no machine-printed digits in the com-

Table 7.4 Summary of error rates on the USPS set. Note that two variants of this database are used in the literature; one of them (denoted by USPS$^+$) is enhanced by a set of machine-printed characters which have been found to improve the test error. Note that the virtual SV systems perform best out of all systems trained on the original USPS set.

Classifier	Training set	Test error	Reference
Linear SVM	USPS	8.9%	[470]
Relevance Vector Machine	USPS	5.1%	Chapter 16
Hard margin SVM	USPS	4.6%	[62]
SVM	USPS	4.0%	[470]
Hyperplane on KPCA features	USPS	4.0%	Chapter 14
KFD	USPS	3.7%	Chapter 15
Virtual SVM	USPS	3.2%	Chapter 11
Virtual SVM, local kernel	USPS	3.0%	Chapter 13
Nearest neighbor	USPS$^+$	5.9%	[496]
LeNet1	USPS$^+$	5.0%	[318]
Local learning Approach	USPS$^+$	3.3%	[65]
Boosted Neural Net	USPS$^+$	2.6%	[148]
Tangent distance	USPS$^+$	2.6%	[496]
Human error rate	—	2.5%	[79]

monly used test set (size 2007), this addition distorts the original learning problem to a situation where results become somewhat hard to interpret. For our experiments, we only had the original 7291 training examples at our disposal. Of all the systems trained on this original set, the SVM system of Chapter 13 performs best.

7.8.2 Universality of the Support Vector Set

In the present section, we report empirical evidence that the SV set contains all the information necessary to solve a given classification task: Using the Support Vector algorithm to train three different types of handwritten digit classifiers, we observe that these types of classifiers construct their decision surface from small, strongly overlapping subsets of the database.

Overlap of SV Sets

To study the Support Vector sets for three different types of SV classifiers, we use the optimal kernel parameters on the USPS set according to Table 7.3. Table 7.5 shows that all three classifiers use around 250 Support Vectors per two-class-classifier (less than 4% of the training set), of which there are 10. The *total* number of different Support Vectors of the ten-class-classifiers is around 1600. It is less than 2500 (10 times the above 250), since for instance a particular vector that has been used as a positive SV (i.e., $y_i = +1$ in (7.25)) for digit 7, might at the same time be a negative SV ($y_i = -1$) for digit 1.

Table 7.6 shows that the SV sets of the different classifiers have about 90% overlap. This surprising result has been reproduced on the MNIST OCR set [467].

Table 7.5 First row: Total number of different SVs in three different ten-class-classifiers (i.e., number of elements of the union of the ten two-class-classifier SV sets), obtained by choosing different functions k in (7.25) and (7.29); Second row: Average number of SVs per two-class-classifier (USPS database size: 7291) (from [470]).

	Polynomial	RBF	Sigmoid
total # of SVs	1677	1498	1611
average # of SVs	274	235	254

Table 7.6 Percentage of the SV set of [column] contained in the SV set of [row]; for ten-class classifiers *(top)*, and binary recognizers for digit class 7 *(bottom)* (USPS set) (from [470]).

	Polynomial	RBF	Sigmoid
Polynomial	100	93	94
RBF	83	100	87
Sigmoid	90	93	100

	Polynomial	RBF	Sigmoid
Polynomial	100	84	93
RBF	89	100	92
Sigmoid	93	86	100

Using a leave-one-out procedure similar to Proposition 7.4, Vapnik and Watkins have put forward a theoretical argument for shared SVs. We state it in the following form: If the SV set of three SV classifiers had no overlap, we could obtain a fourth classifier which has zero test error.

Voting Argument for Shared SVs

To see why this is the case, note that if a pattern is left out of the training set, it will always be classified correctly by voting between the three SV classifiers trained on the remaining examples: Otherwise, it would have been an SV of at least two of them, if kept in the training set. The expectation of the number of patterns which are SVs of at least two of the three classifiers, divided by the training set size, thus forms an upper bound on the expected test error of the voting system. Regarding error rates, it would thus in fact be desirable to be able to construct classifiers with different SV sets. An alternative explanation, studying the effect of the input density on the kernel, was recently proposed by Williams [597]. Finally, we add that the result is also plausible in view of the similar regularization characteristics of the different kernels that were used (see Chapter 4).

Training on SV Sets

As described in Section 7.3, the Support Vector set contains all the information a given classifier needs for constructing the decision function. Due to the overlap in the Support Vector sets of different classifiers, we can even train classifiers on the Support Vector set of *another* classifier; the latter having a different kernel to the former. Table 7.7 shows that this leads to results comparable to those after training

Table 7.7 Training classifiers on the Support Vector sets of other classifiers, leads to performances on the test set (USPS problem) which are as good as the results for training on the full database (number of errors on the 2007-element test set are shown, for two-class classifiers separating digit 7 from the rest). Additionally, the results for training on a random subset of the database of size 200 are displayed.

kernel	trained on: *size:*	poly-SVs *178*	rbf-SVs *189*	tanh-SVs *177*	full db *7291*	rnd. subs. *200*
Poly		13	13	12	13	23
RBF		17	13	17	15	27
tanh		15	13	13	15	25

on the whole database. In Section 11.3, we will use this finding as a motivation for a method to make SVMs transformation invariant, to obtain *virtual SV* machines.

What do these results concerning the nature of Support Vectors tell us? Learning can be viewed as inferring regularities from a set of training examples. Much research has been devoted to the study of various learning algorithms, which allow the extraction of these underlying regularities. No matter how different the outward appearance of these algorithms is, they must all rely on intrinsic regularities of the data. If the learning has been successful, these intrinsic regularities are captured in the values of certain parameters of a learning machine; for a polynomial classifier, these parameters are the coefficients of a polynomial, for a neural network, they are weights, biases, and gains, and for a radial basis function classifier, they are weights, centers, and widths. This variety of different representations of the intrinsic regularities, however, conceals the fact that they all stem from a common root. This is why SVMs with different kernel functions identify the same subset of the training examples as crucial for the regularity to be learned.

7.8.3 Other Applications

SVMs have been successfully applied in other computer vision tasks, which relate to the OCR problems discussed above. Examples include object and face detection and recognition, as well as image retrieval [57, 467, 399, 419, 237, 438, 99, 75].

Another area where SVMs have been used with success is that of *text categorization*. Being a high-dimensional problem, text categorization has been found to be well suited for SVMs. A popular benchmark is the Reuters-22173 text corpus. The news agency Reuters collected 21450 news stories from 1997, and partitioned and indexed them into 135 different categories. The feature typically used to classify Reuters documents are 10^4-dimensional vectors containing word frequencies within a document (sometimes called the "bag-of-words" representation of texts, as it completely discards the information on word ordering). Using this coding, SVMs have led to excellent results, see [155, 265, 267, 150, 333, 542, 149, 326].

Since the use of classification techniques is ubiquitous throughout technology,

we cannot give an exhaustive listing of all successful SVM applications. We thus conclude the list with some of the more exotic applications, such as in High-Energy-Physics [19, 558], in the monitoring of household appliances [390], in protein secondary structure prediction [249], and, with rather intriguing results, in the design of decision feedback equalizers (DFE) in telephony [105].

7.9 Summary

This chapter introduced SV pattern recognition algorithms. The crucial idea is to use kernels to reduce a complex classification task to one that can be solved with separating hyperplanes. We discussed what kind of hyperplane should be constructed in order to get good generalization performance, leading to the idea of large margins. It turns out that the concept of large margins can be justified in a number of different ways, including arguments based on statistical learning theory, and compression schemes. We described in detail how the optimal margin hyperplane can be obtained as the solution of a quadratic programming problem. We started with the linear case, where the hyperplane is constructed in the space of the inputs, and then moved on to the case where we use a kernel function to compute dot products, in order to compute the hyperplane in a feature space.

Two further extensions greatly increase the applicability of the approach. First, to deal with noisy data, we introduced so-called slack variables in the optimization problem. Second, for problems that have more than just two classes, we described a number of generalizations of the binary SV classifiers described initially.

Finally, we reported applications and benchmark comparisons for the widely used USPS handwritten digit task. SVMs turn out to work very well in this field, as well as in a variety of other domains mentioned briefly.

7.10 Problems

7.1 (Weight Vector Scaling •) *Show that instead of the "*1*" on the right hand side of the separation constraint (7.11), we can use any positive number* $\gamma > 0$*, without changing the optimal margin hyperplane solution. What changes in the soft margin case?*

7.2 (Dual Perceptron Algorithm [175] ••) *Kernelize the perceptron algorithm described in footnote 1. Which of the patterns will appear in the expansion of the solution?*

7.3 (Margin of Optimal Margin Hyperplanes [62] ••) *Prove that the geometric margin* ρ *of the optimal margin hyperplane can be computed from the solution* $\boldsymbol{\alpha}$ *via*

$$\rho^{-2} = \sum_{i=1}^{m} \alpha_i. \tag{7.68}$$

we cannot give an exhaustive listing of all successful SVM applications. We thus conclude the list with some of the more exotic applications, such as in High-Energy-Physics [19, 558], in the monitoring of household appliances [390], in protein secondary structure prediction [249], and, with rather intriguing results, in the design of decision feedback equalizers (DFE) in telephony [105].

7.9 Summary

This chapter introduced SV pattern recognition algorithms. The crucial idea is to use kernels to reduce a complex classification task to one that can be solved with separating hyperplanes. We discussed what kind of hyperplane should be constructed in order to get good generalization performance, leading to the idea of large margins. It turns out that the concept of large margins can be justified in a number of different ways, including arguments based on statistical learning theory, and compression schemes. We described in detail how the optimal margin hyperplane can be obtained as the solution of a quadratic programming problem. We started with the linear case, where the hyperplane is constructed in the space of the inputs, and then moved on to the case where we use a kernel function to compute dot products, in order to compute the hyperplane in a feature space.

Two further extensions greatly increase the applicability of the approach. First, to deal with noisy data, we introduced so-called slack variables in the optimization problem. Second, for problems that have more than just two classes, we described a number of generalizations of the binary SV classifiers described initially.

Finally, we reported applications and benchmark comparisons for the widely used USPS handwritten digit task. SVMs turn out to work very well in this field, as well as in a variety of other domains mentioned briefly.

7.10 Problems

7.1 (Weight Vector Scaling •) *Show that instead of the "*1*" on the right hand side of the separation constraint (7.11), we can use any positive number* $\gamma > 0$*, without changing the optimal margin hyperplane solution. What changes in the soft margin case?*

7.2 (Dual Perceptron Algorithm [175] ••) *Kernelize the perceptron algorithm described in footnote 1. Which of the patterns will appear in the expansion of the solution?*

7.3 (Margin of Optimal Margin Hyperplanes [62] ••) *Prove that the geometric margin* ρ *of the optimal margin hyperplane can be computed from the solution* $\boldsymbol{\alpha}$ *via*

$$\rho^{-2} = \sum_{i=1}^{m} \alpha_i. \tag{7.68}$$

B Mathematical Prerequisites

The beginner... should not be discouraged if... he finds that he does not have the prerequisites for reading the prerequisites.

P. Halmos[1]

In this chapter, we introduce mathematical results that might not be known to all readers, but which are sufficiently standard that they not be put into the actual chapters.

This exposition is almost certainly incomplete, and some readers will inevitably happen upon terms in the book that are unknown to them, yet not explained here. Consequently, we also give some further references.

B.1 Probability

B.1.1 Probability Spaces

Let us start with some basic notions of probability theory. For further detail, we refer to [77, 165, 561]. We do not try to be rigorous; instead, we endeavor to give some intuition and explain how these concepts are related to our present interests.

Domain

Assume we are given a nonempty set $\mathcal{X}$, called the *domain* or *universe*. We refer to the elements x of $\mathcal{X}$ as *patterns*. The patterns are generated by a stochastic source. For instance, they could be handwritten digits, which are subject to fluctuations in their generation best modelled probabilistically. In the terms of probability theory, each pattern x is considered the outcome of a *random experiment*.

We would next like to assign probabilities to the patterns. We naively think of a probability as being the limiting frequency of a pattern; in other words, how often, relative to the number of trials, a certain pattern x comes up in a random experiment, if we repeat this experiment infinitely often?

Event
Probability

It turns out to be convenient to be slightly more general, and to talk about the probability of *sets* of possible outcomes; that is, subsets C of $\mathcal{X}$ called *events*. We denote the *probability* that the outcome of the experiment lies in C by

1. Quoted after [429].

$$\mathrm{P}\{x \in C\}. \tag{B.1}$$

If Υ is a logical formula in terms of x, meaning a mapping from $\mathcal{X}$ to $\{\text{true}, \text{false}\}$, then it is sometimes convenient to talk about the probability of Υ being true. We will use the same symbol P in this case, and define its usage as

$$\mathrm{P}\{\Upsilon(x)\} := \mathrm{P}\{x \in C\} \text{ where } C = \{x \in \mathcal{X} | \Upsilon(x) = \text{true}\}. \tag{B.2}$$

Let us also introduce the shorthand

$$\mathrm{P}(C) := \mathrm{P}\{x \in C\}, \tag{B.3}$$

to be read as "the probability of the event C." If P satisfies some fairly natural conditions, it is called a *probability measure*. It is also referred to as the *(probability) distribution of x*.

Distribution of x

In the case where $\mathcal{X} \subset \mathbb{R}^N$, the patterns are usually referred to as *random variables* ($N = 1$) or *random vectors* ($N > 1$). A generic term we shall sometimes use is *random quantity*.[2]

To emphasize the fact that P is the distribution of x, we sometimes denote it as P_x or $\mathrm{P}(x)$.[3] To give the precise definition of a probability measure, we first need to be a bit more formal about which sets C we are going to allow. Certainly,

$$C = \mathcal{X} \tag{B.4}$$

should be a possibility, corresponding to the event that necessarily occurs ("sure thing"). If C is allowed, then its complement,

$$\overline{C} := \mathcal{X} \setminus C, \tag{B.5}$$

should also be allowed. This corresponds to the event "not C." Finally, if $C_1, C_2, \ldots$ are events, then we would like to be able to talk about the probability of the event "C_1 or C_2 or $\ldots$", hence

$$\bigcup_{i=1}^{\infty} C_i \tag{B.6}$$

should be an allowed event.

σ-Algebra

Definition B.1 (σ-Algebra) *A collection $\mathcal{C}$ of subsets of $\mathcal{X}$ is called a* σ-algebra on $\mathcal{X}$ *if*

(i) $\mathcal{X} \in \mathcal{C}$*; in other words, (B.4) is one of its elements;*

(ii) *it is closed under complementation, meaning if $C \in \mathcal{C}$, then also (B.5); and*

(iii) *it is closed under countable*[4] *unions: if $C_1, C_2, \ldots \in \mathcal{C}$, then also (B.6).*

---

2. For simplicity, we are somewhat sloppy in not distinguishing between a random variable and the values it takes. Likewise, we deviate from standard usage in not having introduced random variables as functions on underlying universes of events.
3. The latter is somewhat sloppy, as it suggests that P takes *elements* of $\mathcal{X}$ as inputs, which it does not: P is defined for *subsets* of $\mathcal{X}$.
4. *Countable* means with a number of elements not larger than that of $\mathbb{N}$. Formally, a set

The elements of a σ-algebra are sometimes referred to as measurable sets.

We are now in a position to formalize our intuitions about the probability measure.

Definition B.2 (Probability Measure) *Let $\mathcal{C}$ be a σ-algebra on the domain $\mathcal{X}$. A function*

$$\mathrm{P} : \mathcal{C} \to [0, 1] \tag{B.7}$$

Probability Measure

is called a probability measure *if it is normalized,*

$$\mathrm{P}(\mathcal{X}) = 1, \tag{B.8}$$

and σ-additive, meaning that for sets $C_1, C_2, \ldots \in \mathcal{C}$ that are mutually disjoint ($C_i \cap C_j = \emptyset$ if $i \neq j$), we have

$$\mathrm{P}\left(\bigcup_{i=1}^{\infty} C_i\right) = \sum_{i=1}^{\infty} \mathrm{P}(C_i). \tag{B.9}$$

Measure

As an aside, note that if we drop the normalization condition, we are left with what is called a *measure*.

Probability Space

Taken together, $(\mathcal{X}, \mathcal{C}, \mathrm{P})$ are called a *probability space*. This is the mathematical description of the probabilistic experiment.

B.1.2 IID Samples

Nevertheless, we are not quite there yet, since most of the probabilistic statements in this book do not talk about the outcomes of the experiment described by $(\mathcal{X}, \mathcal{C}, \mathrm{P})$. For instance, when we are trying to learn something about a regularity (that is, about some aspects of P) based on a collection of patterns $x_1, \ldots, x_m \in \mathcal{X}$ (usually called a *sample*), we actually perform the random experiment m times, under identical conditions. This is referred to as *drawing an iid (independent and identically distributed) sample* from P.

Sample

IID Sample

Formally, drawing an iid sample can be described by the probability space $(\mathcal{X}^m, \mathcal{C}^m, \mathrm{P}^m)$. Here, $\mathcal{X}^m$ denotes the m-fold Cartesian product of $\mathcal{X}$ with itself (thus, each element of $\mathcal{X}^m$ is an m-tuple of elements of $\mathcal{X}$), and $\mathcal{C}^m$ denotes the smallest σ-algebra that contains the elements of the m-fold Cartesian product of $\mathcal{C}$ with itself. Likewise, the product measure P^m is determined uniquely by

$$\mathrm{P}^m((C_1, \ldots, C_m)) := \prod_{i=1}^{m} \mathrm{P}(C_i). \tag{B.10}$$

Note that the independence of the "iid" is encoded in (B.10) being a product of measures on $\mathcal{C}$, while the identicality lies in the fact that all the measures on $\mathcal{C}$ are one and the same.

---

is countable if there is a surjective map from $\mathbb{N}$ onto this set; that is, a map with range encompassing the whole set.

By analogy to (B.2), we sometimes talk about the probability of a logical formula involving an m-sample,[5]

$$\mathrm{P}\{Y(x_1, \ldots, x_m)\} := \mathrm{P}^m(\{(x_1, \ldots, x_m) \in \mathcal{X}^m | Y(x_1, \ldots, x_m) = \text{true}\}). \tag{B.11}$$

So far, we have denoted the outcomes of the random experiments as x for simplicity, and have referred to them as patterns. In many cases studied in this book, however, we will not only observe patterns $x \in \mathcal{X}$ but also *targets* $y \in \mathcal{Y}$. For instance, in binary pattern recognition, we have $\mathcal{Y} = \{\pm 1\}$. The underlying regularity is now assumed to generate *examples* (x, y). All of the above applies to this case, with the difference that we now end up with a probability measure on $\mathcal{X} \times \mathcal{Y}$, called the (joint) distribution of (x, y).

B.1.3 Densities and Integrals

We now move on to the concept of a *density*, often confused with the distribution. For simplicity, we restrict ourselves to the case where $\mathcal{X} = \mathbb{R}^N$; in this instance, $\mathcal{C}$ is usually taken to be the *Borel σ-algebra*.[6]

Definition B.3 (Density) *We say that the nonnegative function p is the* density *of the distribution* P *if for all $C \in \mathcal{C}$,*

$$\mathrm{P}(C) = \int_C p(x)dx. \tag{B.12}$$

If such a p exists, it is uniquely determined.[7]

Not all distributions actually *have* a density. To see this, let us consider a distribution that does. If we plug a set of the form $C = \{x\}$ into (B.12), we see that $\mathrm{P}(\{x\}) = 0$; that is, the distribution assigns zero probability to any set of the form $\{x\}$. We infer that only distributions that assign zero probability to individual points can have densities.[8]

It is important to understand the difference between distributions and densities. The distribution takes *sets* of patterns as inputs, and assigns them a probability between 0 and 1. The density takes an individual pattern as its input, and assigns a nonnegative number (possibly larger than 1) to it. Using (B.12), the density can be used to compute the probability of a set C. If the density is a continuous function, and we use a small neighborhood of point x as the set C, then P is approximately

5. Note that there is some sloppiness in the notation: strictly speaking, we should denote this quantity as P^m — usually, however, it can be inferred from the context that we actually mean the m-fold product measure.
6. Readers not familiar with this concept may simply think of it as a collection that contains all "reasonable" subsets of $\mathbb{R}^N$.
7. *Almost everywhere*; in other words, up to a set N with $\mathrm{P}(N) = 0$.
8. In our case, we can show that the distribution P has a density if and only if it is *absolutely continuous* with respect to the Lebesgue measure on $\mathbb{R}^N$, meaning that every set of Lebesgue-measure zero also has P-measure zero.

the size (i.e. , the measure) of the neighborhood times the value of p; in this case, and in this sense, the two quantities are proportional.

Distribution Function

A more fundamental concept, which exists for *every* distribution of a random quantity taking values in $\mathbb{R}^N$, is the *distribution function*,[9]

$$\mathrm{F} : \mathbb{R}^N \to [0,1] \tag{B.13}$$

$$z \mapsto \mathrm{F}(z) = \mathrm{P}\{[x]_1 < [z]_1 \wedge \ldots \wedge [x]_N < [z]_N\}. \tag{B.14}$$

Finally, we need to introduce the notion of an integral with respect to a measure. Consider a function $f : \mathbb{R}^N \to \mathbb{R}$. We denote by

$$\int_C f(x) d\mathrm{P}(x) \tag{B.15}$$

the integral of a function with respect to the distribution (or measure) P, provided that f is *measurable*. For our purposes, the latter means that for every interval $[a,b] \subset \mathbb{R}$, $f^{-1}([a,b])$ (the set of all points in $\mathbb{R}^N$ that get mapped to $[a,b]$) is an element of $\mathcal{C}$. Component-wise extension to vector-valued functions is straightforward.

In the case where P has a density p, (B.15) equals

$$\int_C f(x) p(x) dx, \tag{B.16}$$

which is a standard integral in $\mathbb{R}^N$, weighted by the density function p.

If P does not have a density, we can define the integral by decomposing the range of f into disjoint half-open intervals $[a_i, b_i)$, and computing the measure of each set $f^{-1}([a_i, b_i))$ using P. The contribution of each such set to the integral is determined by multiplying this measure with the function value (on the set), which by construction is in $[a_i, b_i)$. The exact value of the integral is obtained by taking the limit at infinitely small intervals. This construction, which is the basic idea of the Lebesgue integral, does not rely on f being defined on $\mathbb{R}$; it works for general sets $\mathcal{X}$ as long as they are suitably endowed with a measure.

Empirical Measure

Let us consider a special case. If P is the *empirical measure* with respect to $x_1, \ldots, x_m$,[10]

$$\mathrm{P}^m_{\mathrm{emp}}(C) := \frac{|C \cap \{x_1, \ldots, x_m\}|}{m}, \tag{B.17}$$

which represents the fraction of points that lie in C, then the integral takes the form

$$\int_C f(x) d\mathrm{P}^m_{\mathrm{emp}}(x) = \frac{1}{m} \sum_{i=1}^{m} f(x_i). \tag{B.18}$$

As an aside, note that this shows the empirical risk term (1.17) can actually be thought of as an integral, just like the actual risk (1.18).

9. We use $\wedge$ to denote the logical "and" operation, and $[z]_i$ to denote the i^{th}component of z.
10. By $|.|$ we denote the number of elements in a set.

Expectation, Variance, Covariance

If P is a probability distribution (rather than a general measure), then two more special cases of interest are obtained for particular choices of functions f in (B.15). If f is the identity on $\mathbb{R}^N$, we get the *expectation* $\mathbf{E}[x]$. If $f(x) = (x - \mathbf{E}[x])^2$ (on $\mathbb{R}$), we obtain the *variance* of x, denoted by $\mathrm{var}(x)$. In the N-dimensional case, the functions $f_{ij}(x) = (x_i - \mathbf{E}[x_i])(x_j - \mathbf{E}[x_j])$ lead to the *covariance* $\mathrm{cov}(x_i, x_j)$. For a data set $\{x_1, \ldots, x_m\}$, the matrix $(\mathrm{cov}(x_i, x_j))_{ij}$ is called the *covariance matrix*.

B.1.4 Stochastic Processes

A *stochastic process* y on a set $\mathcal{X}$ is a random quantity indexed by $x \in \mathcal{X}$. This means that for every x, we get a random quantity $y(x)$ taking values in $\mathbb{R}$, or more generally, in a *set* $\mathcal{R}$. A stochastic process is characterized by the joint probability distributions of y on arbitrary finite subsets of $\mathcal{X}$; in other words, of $(y(x_1), \ldots, y(x_m))$.[11]

A *Gaussian process* is a stochastic process with the property that for any $\{x_1, \ldots, x_m\} \subset \mathcal{X}$, the random quantities $(y(x_1), \ldots, y(x_m))$ have a joint Gaussian distribution with mean μ and covariance matrix K. The matrix elements K_{ij} are given by a covariance kernel $k(x_i, x_j)$.

When a Gaussian process is used for learning, the *covariance function* $k(x_i, x_j) := \mathrm{cov}(y(x_i), y(x_j))$ essentially plays the same role as the kernel in a SVM. See Section 16.3 and [587, 596] for further information.

B.2 Linear Algebra

B.2.1 Vector Spaces

We move on to basic concepts of linear algebra, which is to say the study of vector spaces. Additional detail can be found in any textbook on linear algebra (e.g., [170]). The feature spaces studied in this book have a rich mathematical structure, which arises from the fact that they allow a number of useful operations to be carried out on their elements: addition, multiplication with scalars, and the product between the elements themselves, called the dot product.

What's so special about these operations? Let us, for a moment, go back to our earlier example (Chapter 1), where we classify sheep. Surely, nobody would come up with the idea of trying to add two sheep, let alone compute their dot product. The set of sheep does not form a vector space; mathematically speaking, it could be argued that it does not have a very rich structure. However, as discussed in Chapter 1 (cf. also Chapter 2), it is possible to *embed* the set of all sheep into a dot product space such that we can think of the dot product as a measure of

11. Note that knowledge of the finite-dimensional distributions (fdds) does not yield complete information on the properties of the sample paths of the stochastic process; two different processes which have the same fdds are known as *versions* of one another.

Expectation, Variance, Covariance

If P is a probability distribution (rather than a general measure), then two more special cases of interest are obtained for particular choices of functions f in (B.15). If f is the identity on $\mathbb{R}^N$, we get the *expectation* $\mathbf{E}[x]$. If $f(x) = (x - \mathbf{E}[x])^2$ (on $\mathbb{R}$), we obtain the *variance* of x, denoted by $\mathrm{var}(x)$. In the N-dimensional case, the functions $f_{ij}(x) = (x_i - \mathbf{E}[x_i])(x_j - \mathbf{E}[x_j])$ lead to the *covariance* $\mathrm{cov}(x_i, x_j)$. For a data set $\{x_1, \ldots, x_m\}$, the matrix $(\mathrm{cov}(x_i, x_j))_{ij}$ is called the *covariance matrix*.

B.1.4 Stochastic Processes

A *stochastic process* y on a set $\mathcal{X}$ is a random quantity indexed by $x \in \mathcal{X}$. This means that for every x, we get a random quantity $y(x)$ taking values in $\mathbb{R}$, or more generally, in a *set* $\mathcal{R}$. A stochastic process is characterized by the joint probability distributions of y on arbitrary finite subsets of $\mathcal{X}$; in other words, of $(y(x_1), \ldots, y(x_m))$.[11]

A *Gaussian process* is a stochastic process with the property that for any $\{x_1, \ldots, x_m\} \subset \mathcal{X}$, the random quantities $(y(x_1), \ldots, y(x_m))$ have a joint Gaussian distribution with mean μ and covariance matrix K. The matrix elements K_{ij} are given by a covariance kernel $k(x_i, x_j)$.

When a Gaussian process is used for learning, the *covariance function* $k(x_i, x_j) := \mathrm{cov}(y(x_i), y(x_j))$ essentially plays the same role as the kernel in a SVM. See Section 16.3 and [587, 596] for further information.

B.2 Linear Algebra

B.2.1 Vector Spaces

We move on to basic concepts of linear algebra, which is to say the study of vector spaces. Additional detail can be found in any textbook on linear algebra (e.g., [170]). The feature spaces studied in this book have a rich mathematical structure, which arises from the fact that they allow a number of useful operations to be carried out on their elements: addition, multiplication with scalars, and the product between the elements themselves, called the dot product.

What's so special about these operations? Let us, for a moment, go back to our earlier example (Chapter 1), where we classify sheep. Surely, nobody would come up with the idea of trying to add two sheep, let alone compute their dot product. The set of sheep does not form a vector space; mathematically speaking, it could be argued that it does not have a very rich structure. However, as discussed in Chapter 1 (cf. also Chapter 2), it is possible to *embed* the set of all sheep into a dot product space such that we can think of the dot product as a measure of

11. Note that knowledge of the finite-dimensional distributions (fdds) does not yield complete information on the properties of the sample paths of the stochastic process; two different processes which have the same fdds are known as *versions* of one another.

the similarity of two sheep. In this space, we can perform the addition of two sheep, multiply sheep with numbers, compute hyperplanes spanned by sheep, and achieve many other things that mathematicians like.

Vector Space

Definition B.4 (Real Vector Space) *A set $\mathcal{H}$ is called a* vector space *(or* linear space*) over $\mathbb{R}$ if addition and scalar multiplication are defined, and satisfy (for all $\mathbf{x}, \mathbf{x}', \mathbf{x}'' \in \mathcal{H}$, and $\lambda, \lambda' \in \mathbb{R}$)*

$$\mathbf{x} + (\mathbf{x}' + \mathbf{x}'') = (\mathbf{x} + \mathbf{x}') + \mathbf{x}'', \tag{B.19}$$

$$\mathbf{x} + \mathbf{x}' = \mathbf{x}' + \mathbf{x} \in \mathcal{H}, \tag{B.20}$$

$$0 \in \mathcal{H},\ \mathbf{x} + 0 = \mathbf{x}, \tag{B.21}$$

$$-\mathbf{x} \in \mathcal{H},\ -\mathbf{x} + \mathbf{x} = 0, \tag{B.22}$$

$$\lambda\mathbf{x} \in \mathcal{H}, \tag{B.23}$$

$$1\mathbf{x} = \mathbf{x}, \tag{B.24}$$

$$\lambda(\lambda'\mathbf{x}) = (\lambda\lambda')\mathbf{x}, \tag{B.25}$$

$$\lambda(\mathbf{x} + \mathbf{x}') = \lambda\mathbf{x} + \lambda\mathbf{x}', \tag{B.26}$$

$$(\lambda + \lambda')\mathbf{x} = \lambda\mathbf{x} + \lambda'\mathbf{x}. \tag{B.27}$$

The first four conditions amount to saying that $(\mathcal{H}, +)$ is a commutative group.[12]

We have restricted ourselves to vector spaces over $\mathbb{R}$. The definition in the complex case is analogous, both here and in most of what follows. Any non-empty subset of $\mathcal{H}$ that is itself a vector space is called a *subspace* of $\mathcal{H}$.

Linear Combination

Among the things we can do in a vector space are *linear combinations*,

$$\sum_{i=1}^{m} \lambda_i \mathbf{x}_i, \text{ where } \lambda_i \in \mathbb{R}, \mathbf{x}_i \in \mathcal{H}, \tag{B.28}$$

Convex Combination

and *convex combinations*,

$$\sum_{i=1}^{m} \lambda_i \mathbf{x}_i, \text{ where } \lambda_i \geq 0, \sum_i \lambda_i = 1, \mathbf{x}_i \in \mathcal{H}. \tag{B.29}$$

Span

The set $\{\sum_{i=1}^{m} \lambda_i \mathbf{x}_i | \lambda_i \in \mathbb{R}\}$ is referred to as the *span* of the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m$.

A set of vectors $\mathbf{x}_i$, chosen such that none of the $\mathbf{x}_i$ can be written as a linear combination of the others, is called *linearly independent*. A set of vectors $\mathbf{x}_i$ that allows us to uniquely write each element of $\mathcal{H}$ as a linear combination is called a

Basis

basis of $\mathcal{H}$. For the uniqueness to hold, the vectors have to be linearly independent. All bases of a vector space $\mathcal{H}$ have the same number of elements, called the

Dimension

dimension of $\mathcal{H}$.

$\mathbb{R}^N$

The standard example of a finite-dimensional vector space is $\mathbb{R}^N$, the space of column vectors $([\mathbf{x}]_1, \ldots, [\mathbf{x}]_N)^\top$, where the $^\top$ denotes the transpose. In $\mathbb{R}^N$,

12. Note that (B.21) and (B.22) should be read as existence statements. For instance, (B.21) states that there exists an element, denoted by 0, with the required property.

addition and scalar multiplication are defined element-wise. The canonical basis of $\mathbb{R}^N$ is $\{\mathbf{e}_1, \dots, \mathbf{e}_N\}$, where for $j = 1, \dots, N$, $[\mathbf{e}_j]_i = \delta_{ij}$. Here δ_{ij} is the Kronecker symbol;

Kronecker δ_{ij}

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases} \tag{B.30}$$

A somewhat more abstract example of a vector space is the space of all real-valued functions on a domain $\mathcal{X}$, denoted by $\mathbb{R}^{\mathcal{X}}$. Here, addition and scalar multiplication are defined by

$$(f + g)(x) := f(x) + g(x), \tag{B.31}$$

$$(\lambda f)(x) := \lambda f(x). \tag{B.32}$$

We shall return to this example below.

Linear Map

Linear algebra is the study of vector spaces and *linear maps* (sometimes called *operators*) between vector spaces. Given two real vector spaces $\mathcal{H}_1$ and $\mathcal{H}_2$, the latter are maps

$$L : \mathcal{H}_1 \to \mathcal{H}_2 \tag{B.33}$$

that for all $\mathbf{x}, \mathbf{x}' \in \mathcal{H}$, $\lambda, \lambda' \in \mathbb{R}$ satisfy

$$L(\lambda \mathbf{x} + \lambda' \mathbf{x}') = \lambda L(\mathbf{x}) + \lambda' L(\mathbf{x}'). \tag{B.34}$$

It is customary to omit the parentheses for linear maps; thus we normally write $L\mathbf{x}$ rather than $L(\mathbf{x})$.

Let us go into more detail, using (for simplicity) the case where $\mathcal{H}_1$ and $\mathcal{H}_2$ are identical, have dimension N, and are written $\mathcal{H}$. Due to (B.34), a linear map L is completely determined by the values it takes on a basis of $\mathcal{H}$. This can be seen by writing an arbitrary input as a linear combination in terms of the basis vectors $\mathbf{e}_j$, and then applying L;

$$L \sum_{j=1}^{N} \lambda_j \mathbf{e}_j = \sum_{j=1}^{N} \lambda_j L \mathbf{e}_j. \tag{B.35}$$

The image of each basis vector, $L\mathbf{e}_j$, is in turn completely determined by its expansion coefficients A_{ij}, $i = 1, \dots, N$;

$$L\mathbf{e}_j = \sum_{i=1}^{N} A_{ij} \mathbf{e}_i. \tag{B.36}$$

Matrix

The coefficients (A_{ij}) form the *matrix* A of L with respect to the basis $\{\mathbf{e}_1, \dots, \mathbf{e}_N\}$. We often think of linear maps as matrices in the first place, and use the same symbol to denote them. The *unit* (or *identity*) matrix is denoted by $\mathbf{1}$. Occasionally, we also use the symbol $\mathbf{1}$ as the identity map on arbitrary sets (rather than vector spaces).

In this book, we assume elementary knowledge of matrix algebra, including the

Matrix Product

matrix product, corresponding to the composition of two linear maps,

$$(AB)_{ij} = \sum_{n=1}^{N} A_{in} B_{nj}, \tag{B.37}$$

Transpose

and the *transpose* $(A^\top)_{ij} := A_{ji}$.

Inverse and Pseudo-Inverse

The *inverse* of a matrix A is written A^{-1} and satisfies $AA^{-1} = A^{-1}A = \mathbf{1}$. The *pseudo-inverse* $A^\dagger$ satisfies $AA^\dagger A = A$. While every matrix has a pseudo-inverse, not all have an inverse. Those which do are called *invertible* or *nonsingular*, and their inverse coincides with the pseudo-inverse. Sometimes, we simply use the notation A^{-1}, and it is understood that we mean the pseudo-inverse whenever A is not invertible.

B.2.2 Norms and Dot Products

Thus far, we have explained the linear structure of spaces such as the feature space induced by a kernel. We now move on to the *metric* structure. To this end, we introduce concepts of length and angles.

Definition B.5 (Norm) *A function* $\|\cdot\| : \mathcal{H} \to \mathbb{R}_0^+$ *that for all* $\mathbf{x}, \mathbf{x}' \in \mathcal{H}$ *and* $\lambda \in \mathbb{R}$ *satisfies*

$$\|\mathbf{x} + \mathbf{x}'\| \leq \|\mathbf{x}\| + \|\mathbf{x}'\|, \tag{B.38}$$

$$\|\lambda \mathbf{x}\| = |\lambda| \|\mathbf{x}\|, \tag{B.39}$$

$$\|\mathbf{x}\| > 0 \text{ if } \mathbf{x} \neq 0, \tag{B.40}$$

Norm

is called a norm on $\mathcal{H}$. *If we replace the "$>$" in (B.40) by "$\geq$," we are left with what is called a* semi-norm.

Metric

Any norm defines a *metric* d via

$$d(\mathbf{x}, \mathbf{x}') := \|\mathbf{x} - \mathbf{x}'\|; \tag{B.41}$$

likewise, any semi-norm defines a *semi-metric*. The (semi-)metric inherits certain properties from the (semi-)norm, in particular the triangle inequality (B.39) and positivity (B.40).

While every norm gives rise to a metric, the converse is not the case. In this sense, the concept of the norm is stronger. Similarly, every *dot product* (to be introduced next) gives rise to a norm, but not vice versa.

Before describing the dot product, we start with a more general concept.

Definition B.6 (Bilinear Form) *A* bilinear form *on a vector space* $\mathcal{H}$ *is a function*

$$Q : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$$
$$(\mathbf{x}, \mathbf{x}') \to Q(\mathbf{x}, \mathbf{x}') \tag{B.42}$$

with the property that for all $\mathbf{x}, \mathbf{x}', \mathbf{x}'' \in \mathcal{H}$ *and all* $\lambda, \lambda' \in \mathbb{R}$, *we have*

$$Q((\lambda \mathbf{x} + \lambda' \mathbf{x}'), \mathbf{x}'') = \lambda Q(\mathbf{x}, \mathbf{x}'') + \lambda' Q(\mathbf{x}', \mathbf{x}''), \tag{B.43}$$

$$Q(\mathbf{x}'', (\lambda \mathbf{x} + \lambda' \mathbf{x}')) = \lambda Q(\mathbf{x}'', \mathbf{x}) + \lambda' Q(\mathbf{x}'', \mathbf{x}'). \tag{B.44}$$

If the bilinear form also satisfies

$$Q(\mathbf{x}, \mathbf{x}') = Q(\mathbf{x}', \mathbf{x}) \tag{B.45}$$

for all $\mathbf{x}, \mathbf{x}' \in \mathcal{H}$*, it is called* symmetric.

Dot Product

Definition B.7 (Dot Product) *A* dot product *on a vector space* $\mathcal{H}$ *is a symmetric bilinear form,*

$$\begin{aligned} \langle ., . \rangle : \mathcal{H} \times \mathcal{H} &\to \mathbb{R} \\ (\mathbf{x}, \mathbf{x}') &\mapsto \langle \mathbf{x}, \mathbf{x}' \rangle , \end{aligned} \tag{B.46}$$

that is strictly positive definite*; in other words, it has the property that for all* $\mathbf{x} \in \mathcal{H}$*,*

$$\langle \mathbf{x}, \mathbf{x} \rangle \geq 0 \text{ with equality only for } \mathbf{x} = 0. \tag{B.47}$$

Definition B.8 (Normed Space and Dot Product Space) *A* normed space *is a vector space endowed with a norm; a* dot product space *(sometimes called* pre-Hilbert space*) is a vector space endowed with a dot product.*

Any dot product defines a corresponding norm via

$$\|\mathbf{x}\| := \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}. \tag{B.48}$$

Cauchy-Schwarz

We now describe the *Cauchy-Schwarz inequality*: For all $\mathbf{x}, \mathbf{x}' \in \mathcal{H}$,

$$| \langle \mathbf{x}, \mathbf{x}' \rangle | \leq \|\mathbf{x}\| \|\mathbf{x}'\|, \tag{B.49}$$

with equality occurring only if $\mathbf{x}$ and $\mathbf{x}'$ are linearly dependent. In some instances, the left hand side can be much smaller than the right hand side. An extreme case

Orthogonality

is when $\mathbf{x}$ and $\mathbf{x}'$ are *orthogonal*, and $\langle \mathbf{x}, \mathbf{x}' \rangle = 0$.

Basis Expansion

One of the most useful constructions possible in dot product spaces are *orthonormal basis expansions*. Suppose $\mathbf{e}_1, \ldots, \mathbf{e}_N$, where $N \in \mathbb{N}$, form an *orthonormal set*; that is, they are mutually orthogonal and have norm 1. If they also form a basis of $\mathcal{H}$, they are called an *orthonormal basis (ONB)*. In this case, any $\mathbf{x} \in \mathcal{H}$ can be written as a linear combination,

$$\mathbf{x} = \sum_{j=1}^{N} \langle \mathbf{x}, \mathbf{e}_j \rangle \, \mathbf{e}_j. \tag{B.50}$$

The standard example of a dot product space is again $\mathbb{R}^N$. We usually employ the canonical dot product,

$$\langle \mathbf{x}, \mathbf{x}' \rangle := \sum_{i=1}^{N} [\mathbf{x}]_i [\mathbf{x}']_i = \mathbf{x}^\top \mathbf{x}', \tag{B.51}$$

and refer to $\mathbb{R}^N$ as the *Euclidean space of dimension* N. Using this dot product and the canonical basis of $\mathbb{R}^N$, each coefficient $\langle \mathbf{x}, \mathbf{e}_j \rangle$ in (B.50) just picks out one entry from the column vector $\mathbf{x}$, thus $\mathbf{x} = \sum_{j=1}^{N} [\mathbf{x}]_j \mathbf{e}_j$.

Pythagorean Theorem

A rather useful result concerning norms arising from dot products is the *Pythagorean Theorem*. In its general form, it reads as follows:

Theorem B.9 (Pythagoras) *If* $\mathbf{e}_1, \ldots, \mathbf{e}_q$ *are orthonormal (they need not form a basis), then*

$$\|\mathbf{x}\|^2 = \sum_{i=1}^{q} \langle \mathbf{x}, \mathbf{e}_i \rangle^2 + \left\| \mathbf{x} - \sum_{i=1}^{q} \langle \mathbf{x}, \mathbf{e}_i \rangle \mathbf{e}_i \right\|^2. \tag{B.52}$$

Now that we have a dot product, we are in a position to summarize a number of useful facts about matrices.

- It can readily be verified that for the canonical dot product, we have

$$\langle \mathbf{x}, A\mathbf{x}' \rangle = \left\langle A^\top \mathbf{x}, \mathbf{x}' \right\rangle \tag{B.53}$$

for all $\mathbf{x}, \mathbf{x}' \in \mathcal{H}$

Symmetric Matrices

- Matrices A such that $A = A^\top$ are called *symmetric*. Due to (B.53), they can be swapped between the two arguments of the canonical dot product without changing its value
- Symmetric matrices A that satisfy

$$\langle \mathbf{x}, A\mathbf{x} \rangle \geq 0 \tag{B.54}$$

for all $\mathbf{x} \in \mathcal{H}$ are called *positive definite* (cf. Remark 2.16 for a note on this terminology)

- Another interesting class of matrices are the *unitary* (or *orthogonal*) matrices. A unitary matrix U is characterized by an inverse U^{-1} that equals its transpose $U^\top$. Unitary matrices thus satisfy

$$\langle U\mathbf{x}, U\mathbf{x}' \rangle = \left\langle U^\top U\mathbf{x}, \mathbf{x}' \right\rangle = \left\langle U^{-1}U\mathbf{x}, \mathbf{x}' \right\rangle = \langle \mathbf{x}, \mathbf{x}' \rangle \tag{B.55}$$

for all $\mathbf{x}, \mathbf{x}' \in \mathcal{H}$; in other words, they leave the canonical dot product invariant

- A final aspect of matrix theory of interest in machine learning is matrix diagonalization. Suppose A is a linear operator. If there exists a basis $\mathbf{v}_1, \ldots, \mathbf{v}_N$ of $\mathcal{H}$ such that for all $i = 1, \ldots, N$,

$$A\mathbf{v}_i = \lambda_i \mathbf{v}_i, \tag{B.56}$$

Eigenvalue and Eigenvector

with $\lambda_i \in \mathbb{R}$, then A can be *diagonalized*: written in the basis $\mathbf{v}_1, \ldots, \mathbf{v}_N$, we have $A_{ij} = 0$ for all $i \neq j$ and $A_{ii} = \lambda_i$ for all i. The coefficients λ_i are called *eigenvalues*, and the $\mathbf{v}_i$ *eigenvectors*, of A

Let us now consider the special case of symmetric matrices. These can always be diagonalized, and their eigenvectors can be chosen to form an orthonormal basis with respect to the canonical dot product. If we form a matrix V with these eigenvectors as columns, then we obtain the diagonal matrix as $VAV^\top$.

Rayleigh's principle states that the smallest eigenvalue $\lambda_{\min}$ coincides with the

minimum of

$$R(\mathbf{v}) := \frac{\langle \mathbf{v}, A\mathbf{v} \rangle}{\langle \mathbf{v}, \mathbf{v} \rangle}. \tag{B.57}$$

The minimizer of R is an eigenvector with eigenvalue $\lambda_{\min}$. Likewise, the largest eigenvalue and its corresponding eigenvector can be found by maximizing R.
Functions $f : I \to \mathbb{R}$, where $I \subset \mathbb{R}$, can be defined on symmetric matrices A with eigenvalues in I. To this end, we diagonalize A and apply f to all diagonal elements (the eigenvalues).
Since a symmetric matrix is positive definite if and only if all its eigenvalues are nonnegative, we may choose $f(x) = \sqrt{x}$ to obtain the unique *square root* $\sqrt{A}$ of a positive definite matrix A.

Many statements about matrices generalize in some form to operators on spaces of arbitrary dimension; for instance, Mercer's theorem (Theorem 2.10) can be viewed as a generalized version of a matrix diagonalization, with eigenvectors (or eigenfunctions) ψ_j satisfying $\int_{\mathcal{X}} k(x, x')\psi_j(x')\, d\mu(x') = \lambda_j \psi_j(x)$.

B.3 Functional Analysis

Functional analysis combines concepts from linear algebra and analysis. Consequently, it is also concerned with questions of convergence and continuity. For a detailed treatment, cf. [429, 306, 112].

Cauchy Sequence

Definition B.10 (Cauchy Sequence) *A sequence* $(\mathbf{x}_i)_i := (\mathbf{x}_i)_{i \in \mathbb{N}} = (\mathbf{x}_1, \mathbf{x}_2, \ldots)$ *in a normed space* $\mathcal{H}$ *is said to be a* Cauchy sequence *if for every* $\epsilon > 0$*, there exists an* $n \in \mathbb{N}$ *such that for all* $n', n'' > n$*, we have* $\|\mathbf{x}_{n'} - \mathbf{x}_{n''}\| < \epsilon$.

A Cauchy sequence is said to converge *to a point* $\mathbf{x} \in \mathcal{H}$ *if* $\|\mathbf{x}_n - \mathbf{x}\| \to 0$ *as* $n \to \infty$.

Banach / Hilbert Space

Definition B.11 (Completeness, Banach Space, Hilbert Space) *A space* $\mathcal{H}$ *is called* complete *if all Cauchy sequences in the space converge.*

A Banach space *is a complete normed space; a* Hilbert space *is a complete dot product space.*

The simplest example of a Hilbert space (and thus also of a Banach space) is again $\mathbb{R}^N$. More interesting Hilbert spaces, however, have *infinite* dimensionality. A number of surprising things can happen in this case. To prevent the nasty ones, we generally assume that the Hilbert spaces we deal with are *separable*,[13] which means that there exists a countable dense subset. A *dense subset* is a set S such that each element of $\mathcal{H}$ is the limit of a sequence in S. Equivalently, the completion of

13. One of the positive side effects of this is that we essentially only have to deal with one Hilbert space: all separable Hilbert spaces are equivalent, in a sense that we won't define presently.

minimum of

$$R(\mathbf{v}) := \frac{\langle \mathbf{v}, A\mathbf{v} \rangle}{\langle \mathbf{v}, \mathbf{v} \rangle}. \tag{B.57}$$

The minimizer of R is an eigenvector with eigenvalue $\lambda_{\min}$. Likewise, the largest eigenvalue and its corresponding eigenvector can be found by maximizing R.
Functions $f : I \to \mathbb{R}$, where $I \subset \mathbb{R}$, can be defined on symmetric matrices A with eigenvalues in I. To this end, we diagonalize A and apply f to all diagonal elements (the eigenvalues).
Since a symmetric matrix is positive definite if and only if all its eigenvalues are nonnegative, we may choose $f(x) = \sqrt{x}$ to obtain the unique *square root* $\sqrt{A}$ of a positive definite matrix A.

Many statements about matrices generalize in some form to operators on spaces of arbitrary dimension; for instance, Mercer's theorem (Theorem 2.10) can be viewed as a generalized version of a matrix diagonalization, with eigenvectors (or eigenfunctions) ψ_j satisfying $\int_{\mathcal{X}} k(x, x')\psi_j(x')\, d\mu(x') = \lambda_j \psi_j(x)$.

B.3 Functional Analysis

Functional analysis combines concepts from linear algebra and analysis. Consequently, it is also concerned with questions of convergence and continuity. For a detailed treatment, cf. [429, 306, 112].

Cauchy Sequence

Definition B.10 (Cauchy Sequence) *A sequence* $(\mathbf{x}_i)_i := (\mathbf{x}_i)_{i \in \mathbb{N}} = (\mathbf{x}_1, \mathbf{x}_2, \ldots)$ *in a normed space* $\mathcal{H}$ *is said to be a* Cauchy sequence *if for every* $\epsilon > 0$*, there exists an* $n \in \mathbb{N}$ *such that for all* $n', n'' > n$*, we have* $\|\mathbf{x}_{n'} - \mathbf{x}_{n''}\| < \epsilon$.

A Cauchy sequence is said to converge *to a point* $\mathbf{x} \in \mathcal{H}$ *if* $\|\mathbf{x}_n - \mathbf{x}\| \to 0$ *as* $n \to \infty$.

Banach / Hilbert Space

Definition B.11 (Completeness, Banach Space, Hilbert Space) *A space* $\mathcal{H}$ *is called* complete *if all Cauchy sequences in the space converge.*

A Banach space *is a complete normed space; a* Hilbert space *is a complete dot product space.*

The simplest example of a Hilbert space (and thus also of a Banach space) is again $\mathbb{R}^N$. More interesting Hilbert spaces, however, have *infinite* dimensionality. A number of surprising things can happen in this case. To prevent the nasty ones, we generally assume that the Hilbert spaces we deal with are *separable*,[13] which means that there exists a countable dense subset. A *dense subset* is a set S such that each element of $\mathcal{H}$ is the limit of a sequence in S. Equivalently, the completion of

13. One of the positive side effects of this is that we essentially only have to deal with one Hilbert space: all separable Hilbert spaces are equivalent, in a sense that we won't define presently.

S equals $\mathcal{H}$. Here, the *completion* $\overline{S}$ is obtained by adding all limit points of Cauchy sequences to the set. [14]

Example B.12 (Hilbert Space of Functions) *Let $C[a,b]$ denote the real-valued continuous functions on the interval $[a,b]$. For $f, g \in C[a,b]$,*

$$\langle f, g \rangle := \int_a^b f(x)g(x)\,dx \tag{B.58}$$

defines a dot product. The completion of $C[a,b]$ in the corresponding norm is the Hilbert space $L_2[a,b]$ of measurable functions[15] *that are square integrable;*

$L_2[a,b]$

$$\int_a^b f(x)^2\,dx < \infty. \tag{B.59}$$

This notion can be generalized to $L_2(\mathbb{R}^N, \mathrm{P})$. Here, P is a Borel measure on $\mathbb{R}^N$, and the dot product is given by

$$\langle f, g \rangle := \int_{\mathbb{R}^N} f(x)g(x)\,d\mathrm{P}(x). \tag{B.60}$$

One of the most useful properties of Hilbert spaces is that as in the case of finite-dimensional vector spaces, it is possible to compute projections. Before stating the theorem, recall that a subset M of $\mathcal{H}$ is called *closed* if every convergent sequence in $\mathcal{H}$ with elements that lie in M also has its limit in M. Any closed subspace of a Hilbert space is itself a Hilbert space.

Projections

Theorem B.13 (Projections in Hilbert Spaces) *Let $\mathcal{H}$ be a Hilbert space and M be a closed subspace. Then every $\mathbf{x} \in \mathcal{H}$ can be written uniquely as $\mathbf{x} = \mathbf{z} + \mathbf{z}^\perp$, where $\mathbf{z} \in M$ and $\langle \mathbf{z}^\perp, \mathbf{t} \rangle = 0$ for all $\mathbf{t} \in M$. The vector $\mathbf{z}$ is the unique element of M minimizing $\|\mathbf{x} - \mathbf{z}\|$; it is called the* projection $P\mathbf{x} := \mathbf{z}$ of $\mathbf{x}$ onto M. *The projection operator P is a linear map.*

Another feature of Hilbert spaces is that they come with a useful generalization of the concept of a basis. Recall that a basis is a set of vectors that allows us to uniquely write each $\mathbf{x}$ as a linear combination. In the context of infinite-dimensional Hilbert spaces, this is quite restrictive (note that linear combinations (B.28) always involve *finitely* many terms) and leads to bases that are not countable. Therefore, we usually work with what is called a *complete orthonormal system* or an *orthonormal basis (ONB)*.[16] Formally, this is defined as an orthonormal set S in a Hilbert space $\mathcal{H}$ with the property that no other nonzero vector in $\mathcal{H}$ is orthogonal to all elements of S.

Orthonormal Basis

14. Note that the completion is denoted by the same symbol as the set complement. Mathematics is full of this kind of symbol overloading, which adds to the challenge.
15. These are not strictly speaking individual functions, but equivalence classes of functions that are allowed to differ on sets of measure zero.
16. These systems are often referred to as *bases* in the context of Hilbert spaces. This is slightly misleading, since they are not bases in the vector space sense.

Gram-Schmidt Orthonormalization

Separable Hilbert spaces possess countable ONBs, which can be constructed using the *Gram-Schmidt* procedure. Suppose $\{\mathbf{v}_i\}_{i\in\Lambda}$ is a linearly independent set of vectors with a span dense in $\mathcal{H}$, with Λ being a countable index set. A countable ONB $\mathbf{e}_1, \mathbf{e}_2, \ldots$ can then be constructed as follows:

$$\begin{aligned}
&\mathbf{e}_1 := \mathbf{v}_1 / \|\mathbf{v}_1\|, \\
&\mathbf{e}_2 := (\mathbf{v}_2 - P_1\mathbf{v}_2) / \|\mathbf{v}_2 - P_1\mathbf{v}_2\|, \\
&\mathbf{e}_3 := (\mathbf{v}_3 - P_2\mathbf{v}_3) / \|\mathbf{v}_3 - P_2\mathbf{v}_3\|, \\
&\quad\vdots \quad\quad \vdots
\end{aligned} \tag{B.61}$$

Here, we use the shorthand P_n for the operator

$$P_n\mathbf{x} := \sum_{i=1}^{n} \langle \mathbf{e}_i, \mathbf{x} \rangle \, \mathbf{e}_i. \tag{B.62}$$

It is easy to show that P_n projects onto the subspace spanned by $\mathbf{e}_1, \ldots, \mathbf{e}_n$.

If the $\mathbf{v}_1, \mathbf{v}_2, \ldots$ are not linearly independent, then it is possible that $\mathbf{v}_{n+1} - P_n\mathbf{v}_{n+1} = 0$, which means $\mathbf{v}_{n+1}$ can be expressed as a linear combination of $\mathbf{v}_1, \ldots, \mathbf{v}_n$. In this case, we simply leave out $\mathbf{v}_{n+1}$ and proceed with $\mathbf{v}_{n+2}$, shifting all subsequent indices by 1.

ONB Expansion

Using an ONB, we can give basis expansions in infinite-dimensional Hilbert spaces, which look just like basis expansions in the finite-dimensional case. For separable Hilbert spaces, the index set Λ is countable.

Theorem B.14 (ONB Expansions & Parseval's Relation) *Let $\{\mathbf{e}_i\}_{i\in\Lambda}$ be an ONB of the Hilbert space $\mathcal{H}$. Then for each $\mathbf{x} \in \mathcal{H}$,*

$$\mathbf{x} = \sum_{i\in\Lambda} \langle \mathbf{e}_i, \mathbf{x} \rangle \, \mathbf{e}_i \tag{B.63}$$

and

$$\|\mathbf{x}\|^2 = \sum_{i\in\Lambda} \langle \mathbf{e}_i, \mathbf{x} \rangle^2 . \tag{B.64}$$

Note that this generalizes the Pythagorean Theorem to the infinite-dimensional case.

Let us describe an application of this result, with the dual purpose of demonstrating a standard trick from functional analysis, and mathematically justifying a crucial step in the "kernelization" of many algorithms. In Kernel PCA, we need to solve an eigenvalue problem of the form (cf. (14.7))

Kernel PCA

$$\lambda\mathbf{v} = \mathbf{C}\mathbf{v}, \tag{B.65}$$

and we know a priori that all solutions $\mathbf{v}$ lie in the span of $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathcal{H}$. In Chapter 14, we argued that this means we may instead consider the set of equations

$$\langle \mathbf{x}_n, \mathbf{v}_1 \rangle = \langle \mathbf{x}_n, \mathbf{v}_2 \rangle \;\text{ for all } n = 1, \ldots, m, \tag{B.66}$$

where we use the shorthand $\mathbf{v}_1 = \lambda\mathbf{v}$ and $\mathbf{v}_2 = \mathbf{C}\mathbf{v}$.

We are now in a position to prove this formally. It suffices to consider the case where the $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ are orthonormal. If they are not, we first apply the Gram-Schmidt procedure to construct an orthonormal set $\{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$. The latter is a basis for the span of the $\mathbf{x}_i$, hence each $\mathbf{x}_i$ can be written as a linear combination of the $\mathbf{e}_i$. Conversely, each $\mathbf{e}_i$ can be written as a linear combination of the $\mathbf{x}_i$ by construction (cf. (B.61)). Therefore, (B.66) is actually equivalent to the corresponding statement for the orthonormal set $\{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$.

In the orthonormal case, the Parseval relation (B.64), applied to the completion of the span of the $\mathbf{x}_i$ (which is a Hilbert space), implies that (we replace $x = \mathbf{v}_1 - \mathbf{v}_2$)

$$\|\mathbf{v}_1 - \mathbf{v}_2\|^2 = \sum_{i=1}^{m} (\langle \mathbf{x}_i, \mathbf{v}_1 \rangle - \langle \mathbf{x}_i, \mathbf{v}_2 \rangle)^2. \tag{B.67}$$

Therefore $\mathbf{v}_1 = \mathbf{v}_2$ if and only if (14.8) holds true. In a nutshell, the ONB expansion coefficients are unique and completely characterize each vector in a Hilbert space.

We next revisit the L_2 space. Since we will be using the complex exponential, we consider for a moment the case where $\mathcal{H}$ is a Hilbert space over $\mathbb{C}$ rather than $\mathbb{R}$.

Fourier Series

Example B.15 (Fourier Series) *The collection of functions*[17]

$$\left\{ \frac{e^{ix}}{\sqrt{2\pi}}, \frac{e^{-ix}}{\sqrt{2\pi}}, \frac{e^{2ix}}{\sqrt{2\pi}}, \frac{e^{-2ix}}{\sqrt{2\pi}}, \ldots \right\} \tag{B.68}$$

is an ONB for $L_2[0, 2\pi]$. *As a consequence of Theorem B.14, we can thus expand any* $f \in L_2[0, 2\pi]$ *as*

$$\lim_{M \to \infty} \frac{1}{\sqrt{2\pi}} \sum_{n=-M}^{M} c_n e^{inx}, \tag{B.69}$$

where the Fourier coefficients c_n *are given by*[18]

$$c_n = \frac{1}{\sqrt{2\pi}} \int_0^{2\pi} e^{-inx} f(x)\, dx. \tag{B.70}$$

B.3.1 Advanced Topics

ℓ_p^N Spaces

We now move on to concepts that are only used in a few of the chapters; these mainly comprise results that build on [606]. We define normed spaces ℓ_p^N as follows: As vector spaces, they are identical to $\mathbb{R}^N$, but are endowed in addition with p-norms. For $1 \leq p < \infty$, these p-norms are defined as

$$\|x\|_{\ell_p^N} := \|x\|_p = \left(\sum_{j=1}^{N} |x_j|^p \right)^{1/p}; \tag{B.71}$$

17. Here i is the imaginary unit $\sqrt{-1}$.
18. Comparing this to (B.60), we note that there is an unexpected minus sign in e^{-inx}. This is due to the fact that in the complex case, the dot product (B.60) includes a complex conjugation in the first argument.

for $p = \infty$, as

$$\|x\|_{\ell_\infty^N} := \|x\|_\infty = \max_{j=1,\dots,N} |x_j|. \tag{B.72}$$

We use the shorthand ℓ_p to denote the case where $N = \infty$. In this case, it is understood that ℓ_p contains all sequences with finite p-norm. For $N = \infty$, the max in (B.72) is replaced by a sup.

Often the above notations are also used in the case where $0 < p < 1$. In that case, however, we are no longer dealing with norms.

Suppose $\mathcal{F}$ is a class of functions $f : \mathcal{X} \to \mathbb{R}$. The ℓ_∞^N norm of $f \in \mathcal{F}$ *with respect to a sample* $X = (x_1, \dots, x_m)$ is defined as

$$\|f\|_{\ell_\infty^X} := \max_{i=1,\dots,m} |f(x_i)|. \tag{B.73}$$

Likewise,

$$\|f\|_{\ell_p^X} = \|(f(x_1), \dots, f(x_m)\|_{\ell_p^m}. \tag{B.74}$$

L_p Spaces

Given some set $\mathcal{X}$ with a σ-algebra, a measure μ on $\mathcal{X}$, some p in the range $1 \le p < \infty$, and a function $f : \mathcal{X} \to \mathbb{R}$, we define

$$\|f\|_{L_p(\mathcal{X})} := \|f\|_p := \left(\int |f(x)|^p d\mu(x) \right)^{1/p} \tag{B.75}$$

if the integral exists, and

$$\|f\|_{L_\infty(\mathcal{X})} := \|f\|_\infty := \operatorname*{ess\,sup}_{x \in \mathcal{X}} |f(x)|. \tag{B.76}$$

Here, ess sup denotes the essential supremum; that is, the smallest number that upper bounds $|f(x)|$ almost everywhere.

For $1 \le p \le \infty$, we define

$$L_p(\mathcal{X}) := \{ f : \mathcal{X} \to \mathbb{R} \mid \|f\|_{L_p(\mathcal{X})} < \infty \}. \tag{B.77}$$

Here, we have glossed over some details: in fact, these spaces do not consist of functions, but of equivalence classes of functions differing on sets of measure zero. An interesting exception to this rule are reproducing kernel Hilbert spaces (Section 2.2.3). For these, we know that point evaluation of all functions in the space is well-defined: it is determined by the reproducing kernel, see (2.29).

Let $\mathfrak{L}(E, G)$ be the set of all bounded linear operators T between the normed spaces $(E, \|\cdot\|_E)$ and $(G, \|\cdot\|_G)$; in other words, operators such that the image of the (closed) unit ball,

$$U_E := \{ x \in E \mid \|x\|_E \le 1 \}, \tag{B.78}$$

is bounded. The smallest such bound is called the *operator norm*,

$$\|T\| := \sup_{x \in U_E} \|Tx\|_G. \tag{B.79}$$