Learning with Kernels

Learning with Kernels

by
Bernhard Schölkopf
Alexander J. Smola

The MIT Press
Cambridge, Massachusetts
London, England

Printed and bound in the United States of America

Library of Congress Cataloging-in-Publication Data

Learning with Kernels / by Bernhard Schölkopf,
Alexander J. Smola.
p. cm.
Includes bibliographical references and index.
ISBN 0-xxx-xxxxx-x (alk. paper)
1. Machine learning. 2. Algorithms. 3. Kernel functions
I. Schölkopf, Bernhard. II. Smola, Alexander J.
xxxx.x.xxx 2000
xxx.x'x–xxxx 00.xxxxx
CIP

Contents

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 of these concepts is that of a kernel. Rather than immediately going 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 that, we report some basic insights from statistical learning theory, the mathematical theory that underlies the basic idea of SV learning (Section 1.3). Finally, we briefly review some of the main kernel algorithms, namely SV machines (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

Training Data

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

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

Here, $\mathcal{X}$ is some nonempty set that the *patterns* x_i (sometimes called *cases* or *inputs*) are taken from, sometimes referred to as the *domain*; the y_i are called *labels*, *targets*, or *outputs*. Note that there are only two classes of patterns. For the sake of mathematical convenience, they are labeled 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 kind 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 $y \in \{\pm 1\}$.[1] 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 **??**), the function k is called a *kernel* [19, 1, 5, 6, 16].

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

Dot Product

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

Here, $[\mathbf{x}]_i$ denotes the i-th 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 **??**, we give a general definition of dot products. Usually, however, it is sufficient to think of dot products as (1.3).

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

Length

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

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

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

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

Feature Space

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 live 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 **??**, where the theory of kernels is developed in some detail.

Presently, we will give an example of a kernel algorithm.

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 $\mathcal{H}$.[2] Using the dot product, we can measure distances in that space. The basic idea of the algorithm will be to assign a previously unseen pattern to the class whose mean is closer.

2. For the definition of a dot product space, see Section **??**.

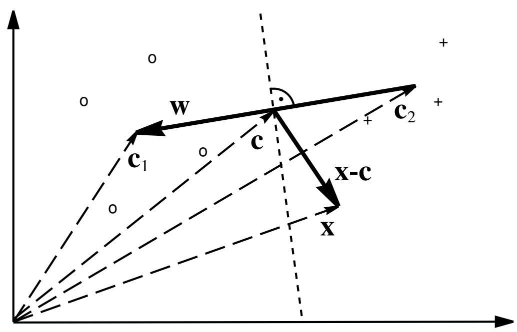

Figure 1.1 A simple geometric classification algorithm: given two classes of points (depicted by 'o' and '+'), compute their means $\mathbf{c}_1, \mathbf{c}_2$ and assign a test pattern $\mathbf{x}$ to the class whose mean it is closer to. This can be done by looking at the dot product between $\mathbf{x} - \mathbf{c}$ (where $\mathbf{c} = (\mathbf{c}_1 + \mathbf{c}_2)/2$) and $\mathbf{w} := \mathbf{c}_1 - \mathbf{c}_2$, 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 $\mathbf{w}$.

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

$$\mathbf{c}_1 = \frac{1}{m_1} \sum_{\{i : y_i = +1\}} \mathbf{x}_i, \tag{1.7}$$

$$\mathbf{c}_2 = \frac{1}{m_2} \sum_{\{i : y_i = -1\}} \mathbf{x}_i, \tag{1.8}$$

where m_1 and m_2 are the number of examples with positive and negative labels, respectively. We assume that both classes are non-empty, that is, $m_1, m_2 > 0$. We then assign a new point $\mathbf{x}$ to the class whose mean is closer to it (Figure 1.1). This geometric construction can be formulated in terms of the dot product $\langle \cdot, \cdot \rangle$. Half-way in between $\mathbf{c}_1$ and $\mathbf{c}_2$ lies the point $\mathbf{c} := (\mathbf{c}_1 + \mathbf{c}_2)/2$. 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 $\mathbf{w} := \mathbf{c}_1 - \mathbf{c}_2$ connecting the class means. This leads to

$$\begin{aligned} y &= \operatorname{sgn} \langle (\mathbf{x} - \mathbf{c}), \mathbf{w} \rangle \\ &= \operatorname{sgn} \langle (\mathbf{x} - (\mathbf{c}_1 + \mathbf{c}_2)/2), (\mathbf{c}_1 - \mathbf{c}_2) \rangle \\ &= \operatorname{sgn} (\langle \mathbf{x}, \mathbf{c}_1 \rangle - \langle \mathbf{x}, \mathbf{c}_2 \rangle + b). \end{aligned} \tag{1.9}$$

Here, we have defined the offset

$$b := \frac{1}{2}(\|\mathbf{c}_2\|^2 - \|\mathbf{c}_1\|^2), \tag{1.10}$$

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 that can be written as a linear equation.

It will prove 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 first 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*

$$\begin{aligned} y &= \operatorname{sgn}\left(\frac{1}{m_1} \sum_{\{i:y_i=+1\}} \langle \mathbf{x}, \mathbf{x}_i \rangle - \frac{1}{m_2} \sum_{\{i:y_i=-1\}} \langle \mathbf{x}, \mathbf{x}_i \rangle + b \right) \\ &= \operatorname{sgn}\left(\frac{1}{m_1} \sum_{\{i:y_i=+1\}} k(x, x_i) - \frac{1}{m_2} \sum_{\{i:y_i=-1\}} k(x, x_i) + b \right). \end{aligned} \tag{1.11}$$

Similarly, the offset becomes

$$b := \frac{1}{2} \left(\frac{1}{m_2^2} \sum_{\{(i,j):y_i=y_j=-1\}} k(x_i, x_j) - \frac{1}{m_1^2} \sum_{\{(i,j):y_i=y_j=+1\}} k(x_i, x_j) \right). \tag{1.12}$$

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 $b = 0$), 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 integral one,[3]

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

In that 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

Parzen Windows

by the *Parzen windows* estimators of the two class densities,

$$p_1(x) := \frac{1}{m_1} \sum_{\{i:y_i=+1\}} k(x, x_i), \tag{1.14}$$

$$p_2(x) := \frac{1}{m_2} \sum_{\{i:y_i=-1\}} k(x, x_i), \tag{1.15}$$

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

Given some point x, the label is then simply computed by checking which of the two values, $p_1(x)$ or $p_2(x)$, is larger, which directly leads 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

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

with in detail. Both take the form of kernel expansions on the input domain,

$$y = \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_i k(x, x_i) + b\right). \tag{1.16}$$

In both cases, the expansions correspond to separating hyperplanes in a feature space. Both are example-based in the sense that the kernels are centered on the training patterns, that is, one of the two arguments of the kernels is always a training pattern. A test point is classified by comparing it to all the training points that appear in (1.16) with a nonzero weight.

The main point where the more sophisticated techniques to be discussed in the remainder of the book will deviate from (1.11) is in the selection of the patterns that the kernels are centered on, that is, in the weights α_i that are put 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 presently, cf. (1.11), the weights were either $(1/m_1)$ or $(-1/m_2)$, depending on which class the pattern belonged to.

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 [34, 13, 14]. This will allow us to indicate the factors affecting the design of "better" algorithms. Rather than just provising tools to come up with new algorithms, we thus 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

$$f : \mathcal{X} \to \{\pm 1\} \tag{1.17}$$

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 equivalently plot the boundaries where it switches between

1 and -1, which is what do presently. 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. The picture in the middle, finally, represents a compromise, by using a model with an intermediate complexity, which gets most points right, without putting too much trust in anhy individual point.

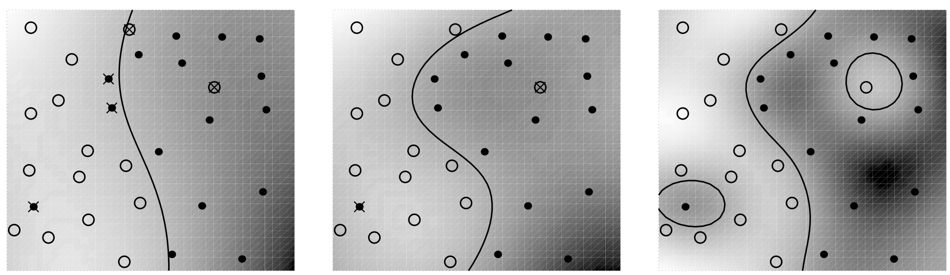

Figure 1.2 2D toy example of a binary classification, solved by three models (shown are the decision boundaries). The models vary in complexity, ranging from a simple one *(left)*, which misclassifies a large number of points, to a complex one *(right)*, which "trusts" each point and comes up with solution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

The goal of statistical learning theory is to place these handwaving arguments in a mathematical framework.

We assume that the data are generated independently from some unknown (but fixed) probability distribution $\mathrm{P}(x, y)$.[4] 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 an f that will correctly classify unseen examples (x, y), that is, we want $f(x) = y$ for examples (x, y) that are also generated from $\mathrm{P}(x, y)$.[5] Correctness of the classification is measured by means of the *zero-one*

4. For a definition of a probability distribution, see Section **??**.
5. We are mostly using the term *example* to denote a pair consisting of a training pattern x and the corresponding target y.

loss function $\frac{1}{2}|f(x)-y|$. Note that the loss is 0 if (x,y) is classified correctly, and 1 otherwise.

If we put no restriction on the set of functions that we choose our estimated f from, however, even a function that does very well on the training data, e.g., by satisfying $f(x_i) = y_i$ for all $i = 1,\ldots,m$, need not generalize well to unseen examples. To see this, note that for each function f and any test set $(\bar{x}_1,\bar{y}_1),\ldots,(\bar{x}_{\bar{m}},\bar{y}_{\bar{m}}) \in \mathcal{X}\times\{\pm1\}$, 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 $i = 1,\ldots,\bar{m}$. 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 only minimizing the (average) *training error* (or *empirical risk*),

Test Data

Empirical Risk

$$R_{emp}[f] = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}|f(x_i)-y_i|, \tag{1.18}$$

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

Risk

$$R[f] = \int \frac{1}{2}|f(x)-y|\; d\mathrm{P}(x,y). \tag{1.19}$$

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.

Statistical learning theory (Chapter **??**, [39, 34, 35, 12, 36, 3]), or VC (Vapnik-Chervonenkis) theory, shows that it is imperative to restrict the set of functions that f is chosen from to one which has a *capacity* that is 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* [34].

Capacity

VC dimension

The best-known capacity concept of VC theory is the *VC dimension*, defined as follows: each function of the class labels the training patterns in a certain way. Since the labels are in $\{\pm1\}$, there are at most 2^m different labelings for m patterns. However, a given class of functions might not be sufficiently rich to induce *all* these labelings; in other words, it might not be able 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. As such, it is necessarily somewhat crude. Examples of more accurate capacities 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* [17, 2]. For further details, cf. Chapters **??** and **??**.

Whilst it will be difficult for the non-expert to appreciate the results of VC theory

VC Bound

already in this chapter, we will nevertheless briefly describe an example of a VC bound is the following: 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, with a probability of at least $1 - \delta$ over the drawing of the training sample,[6] the bound

$$R[f] \leq R_{emp}[f] + \phi\left(\frac{h}{m}, \frac{\log(\delta)}{m}\right) \tag{1.20}$$

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

$$\phi\left(\frac{h}{m}, \frac{\log(\delta)}{m}\right) = \sqrt{\frac{h\left(\log\frac{2m}{h} + 1\right) - \log(\delta/4)}{m}}. \tag{1.21}$$

The bound (1.20) deserves further explanatory remarks. 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 labelings by correctly separating all training examples, however, this machine will necessarily require a large VC dimension h. Therefore, the confidence term (1.21), increasing monotonically with h, will be large, and the bound (1.20) will *not* support possible hopes that due to the small training error, we should expect a small test error. This makes it understandable how it can hold independent of assumptions about the underlying distribution $\mathrm{P}(x, y)$: it always holds (provided that $h < m$), but it does not always make a nontrivial prediction. It is a bound on an error rate (which necessarily lies in the interval $[0, 1]$), and thus it becomes meaningless if it is larger than 1. In order to get nontrivial predictions from (1.20), 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 $\mathrm{P}(x, y)$. 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

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

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 previously). 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. [41, 38] considered the class of hyperplanes in some dot product space $\mathcal{H}$,

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = 0 \quad \mathbf{w} \in \mathcal{H}, b \in R, \tag{1.22}$$

corresponding to decision functions

$$f(\mathbf{x}) = \operatorname{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle + b), \tag{1.23}$$

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 **??**), among all hyperplanes separating the data, there exists a unique one, called the *optimal hyperplane*, distinguished by the maximum margin of separation between any training point and the hyperplane,

Optimal Hyperplane

$$\max_{\mathbf{w},b} \; \min\{\|\mathbf{x} - \mathbf{x}_i\| : \mathbf{x} \in \mathcal{H}, \langle \mathbf{w}, \mathbf{x} \rangle + b = 0, i = 1, \ldots, m\}. \tag{1.24}$$

Second (see Chapter **??**), 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 ([39, 34, 43, 4], cf. Chapters **??**, **??**, **??**). In addition, it is *computationally* attractive, since we will show below that it can be constructed by solving a quadratic programming problem for which there exist efficient algorithms (see Chapters **??** and **??**).

Note that the form of the decision function 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 $\mathbf{w} = \mathbf{c}_1 - \mathbf{c}_2$.

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, one has to compute

$$\min_{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}} \tau(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 \tag{1.25}$$

$$\text{subject to} \quad y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1, \quad i = 1, \ldots, m. \tag{1.26}$$

Note that the constraints (1.26) 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 (1.26). 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 $(\mathbf{w}, b)$ were the solution. Let us rescale it by multiplication with some $0 < \lambda < 1$. Since $\lambda > 0$, the constraints are still satisfied.

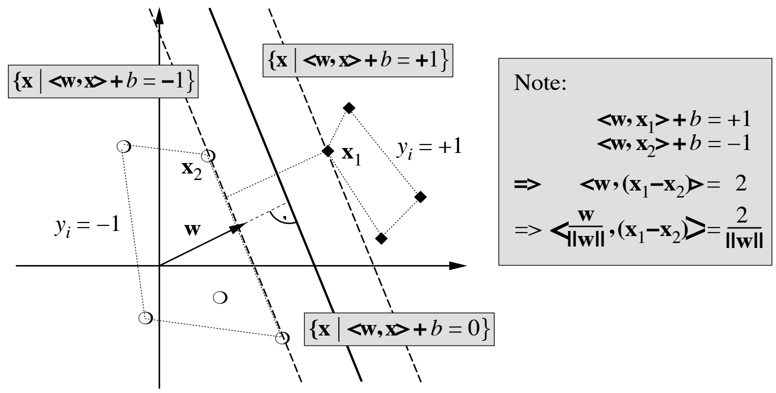

Figure 1.3 A binary classification toy problem: separate balls from diamonds. The *optimal hyperplane* (1.24) is shown as a solid line. The problem being separable, there exists a weight vector $\mathbf{w}$ and a threshold b such that $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 0$ $(i = 1, \ldots, m)$. 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*, measured perpendicularly to the hyperplane, equals $2/\|\mathbf{w}\|$. 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 $\mathbf{w}/\|\mathbf{w}\|$.

However, since $\lambda < 1$, the length of $\mathbf{w}$ has decreased. Hence $(\mathbf{w}, b)$ was not the minimizer in the first place.

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 $\mathbf{w}$, (1.25). If $\|\mathbf{w}\|$ were 1, then the left hand side of (1.26) would equal the distance of $\mathbf{x}_i$ to the hyperplane (cf. (1.24)). In general, we have to divide it by $\|\mathbf{w}\|$ to transform it into the 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 maximal.

A more detailed explanation why this leads to the maximum margin hyperplane will be given in Chapter **??**. A short summary of the argument is also given in Figure 1.3.

The function τ in (1.25) is called the *objective function*, while (1.26) are called *inequality constraints*. Together, they form a so-called *constrained optimization problem*. Problems of this kind are dealt with by introducing *Lagrange multipliers*

Lagrangian

$\alpha_i \geq 0$ and a *Lagrangian*[7]

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i \left(y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1 \right). \tag{1.27}$$

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

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 (1.26) 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 $\alpha_i\,(y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - 1)$ 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 **??**).

KKT Conditions

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

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \quad \frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \tag{1.28}$$

leads to

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{1.29}$$

and

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i. \tag{1.30}$$

The solution vector thus has an expansion in terms of a subset of the training patterns, namely those patterns whose α_i is non-zero, called *support vectors (SVs)* (cf. (1.16) in the initial example). By the KKT conditions

Support Vector

$$\alpha_i [y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1] = 0, \qquad i = 1, \ldots, m, \tag{1.31}$$

the SVs lie on the margin (cf. Figure 1.3). 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. (1.26)) does not play a role in the optimization, and they do not appear in the expansion (1.30). This nicely captures our intuition of the problem: as the hyperplane (cf. Figure 1.3) is completely determined by the patterns closest to it, the solution should not depend on the other examples.

By substituting (1.29) and (1.30) into the Lagrangian (1.27), 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:

Dual Problem

$$\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{1.32}$$

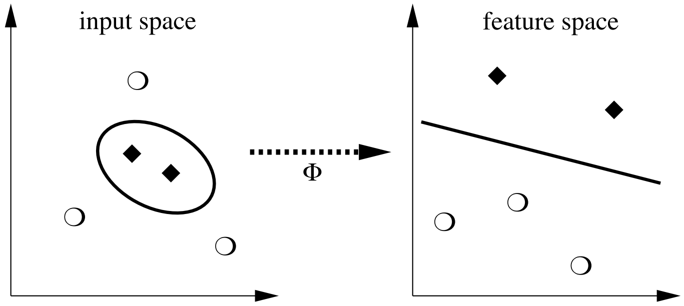

Figure 1.4 The idea of SV machines: map the training data into a higher-dimensional feature space via Φ, and construct a separating hyperplane with maximum margin there. This yields a nonlinear decision boundary in input space. By the use of a kernel function (1.2), it is possible to compute the separating hyperplane without explicitly carrying out the map into the feature space.

$$\text{subject to} \qquad \alpha_i \geq 0, \quad i = 1, \dots, m, \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{1.33}$$

Using (1.30), the hyperplane decision function (1.23) can thus be written as

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b\right) \tag{1.34}$$

where b is computed by exploiting (1.31) (for details, cf. Chapter **??**).

The structure of the optimization problem closely resembles those that typically arise in Lagrange's formulation of mechanics (e.g., [15]). There, often only a subset of constraints become active, too. 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, those walls could just as well be removed.

Seen in this light, it is not too surprising that it is possible to give a mechanical interpretation of optimal margin hyperplanes [8]: If we assume that each SV $\mathbf{x}_i$ exerts a perpendicular force of size α_i and sign y_i on a solid plane sheet lying along the hyperplane, then the solution satisfies the requirements of mechanical stability. The constraint (1.29) states that the forces on the sheet sum to zero; and (1.30) implies that the torques also sum to zero, via $\sum_i \mathbf{x}_i \times y_i \alpha_i \mathbf{w}/\|\mathbf{w}\| = \mathbf{w} \times \mathbf{w}/\|\mathbf{w}\| = 0$.[8]

1.5 Support Vector Classification

We now have all the tools to describe SV machines (Figure 1.4). Everything in the last section was formulated in a dot product space. We think of this space as the

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

feature space $\mathcal{H}$ described in Section 1.1. To express the formulas in terms of the input patterns living 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',

$$k(x, x') = \langle \mathbf{x}, \mathbf{x}' \rangle . \tag{1.35}$$

This substitution, which is sometimes referred to as the *kernel trick*, was used by Boser, Guyon, and Vapnik [6] to extend the *Generalized Portrait* hyperplane classifier of Vapnik and co-workers [41, 39] to nonlinear Support Vector machines. Aizerman et al [1] called $\mathcal{H}$ the *linearization space*, and used 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. The weight vector (cf. (1.30)) then becomes an expansion in feature space, and therefore will typically no longer correspond to the Φ-image of a single vector from input space (cf. Chapter **??**). We thus obtain decision functions of the form (cf. (1.34))

Decision Function

$$\begin{aligned} f(x) &= \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i \langle \Phi(x), \Phi(x_i) \rangle + b \right) \\ &= \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i k(x, x_i) + b \right), \end{aligned} \tag{1.36}$$

and the following quadratic program (cf. (1.32)):

$$\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \tag{1.37}$$

$$\text{subject to } \alpha_i \geq 0, \quad i = 1, \ldots, m, \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{1.38}$$

Figure 1.5 shows an example of this approach, using a Gaussian radial basis function kernel. We will study the different possibilities for the kernel function in detail below (Chapters **??** and Chapter **??**).

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 (1.26), one introduces slack variables [9, 35, 28]

Soft Margin Hyperplane

$$\xi_i \geq 0, \quad i = 1, \ldots, m \tag{1.39}$$

in order to relax the constraints (1.26) to

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m. \tag{1.40}$$

A classifier which 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.

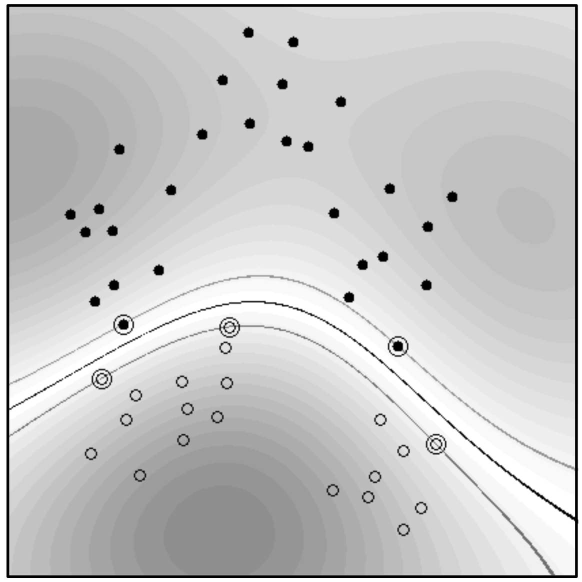

Figure 1.5 Example of an SV classifier found by using a radial basis function kernel $k(x, x') = \exp(-\|x - x'\|^2)$ (here, the input space is $\mathcal{X} = [-1, 1]^2$). Circles and disks are two classes of training examples; the middle line is the decision surface; the outer lines precisely meet the constraint (1.26). 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. Grey values code $|\sum_{i=1}^m y_i \alpha_i k(x, x_i) + b|$, that is, the modulus of the argument of the decision function (1.36). The top and the bottom lines indicate places where it takes the value 1, as enforced by the separation constraints (from [26]).

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

$$\tau(\mathbf{w}, \boldsymbol{\xi}) = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \xi_i \tag{1.41}$$

subject to the constraints (1.39) and (1.40), where the constant $C > 0$ determines the trade-off between margin maximization and training error minimization. Incorporating a kernel, and rewriting it in terms of Lagrange multipliers, this again leads to the problem of maximizing (1.37), subject to the constraints

$$0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m, \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{1.42}$$

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 (1.36). 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

$$\sum_{j=1}^{m} \alpha_j y_j k(x_i, x_j) + b = y_i. \tag{1.43}$$

Geometrically speaking, choosing b amounts to shifting the hyperplane, and (1.43) states that we have to shift the hyperplane such that the SVs with zero slack variables lie on the ± 1 lines of Figure 1.3.

Another possible realization of a soft margin variant of the optimal hyperplane uses the more natural ν-parameterization. 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 come to lie on the wrong side of the hyperplane, 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. (1.41)), and separation constraints that involve a margin parameter ρ,

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq \rho - \xi_i, \quad i = 1, \ldots, m, \tag{1.44}$$

which itself is a variable of the optimization problem. The dual can be shown to consist of maximizing the quadratic part of (1.37), 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 **??**.

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 that case, an analog of the soft margin is constructed in the space of the target values y (note that we now have $y \in \mathbb{R}$) by using Vapnik's *ε-insensitive loss function* [35] (Figure 1.6, for details, see Chapters **??** and **??**) . It quantifies the loss incurred by predicting $f(\mathbf{x})$ instead of y as

ε-Insensitive Loss

$$|y - f(\mathbf{x})|_\varepsilon = \max\{0, |y - f(\mathbf{x})| - \varepsilon\}. \tag{1.45}$$

To estimate a linear regression

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b \tag{1.46}$$

one minimizes

$$\frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} |y_i - f(\mathbf{x}_i)|_\varepsilon. \tag{1.47}$$

Note that the term $\|\mathbf{w}\|^2$ is the same as in pattern recognition (cf. (1.41)); for further details, cf. Chapter **??**.

We can transform this into a constrained optimization problem by introducing,

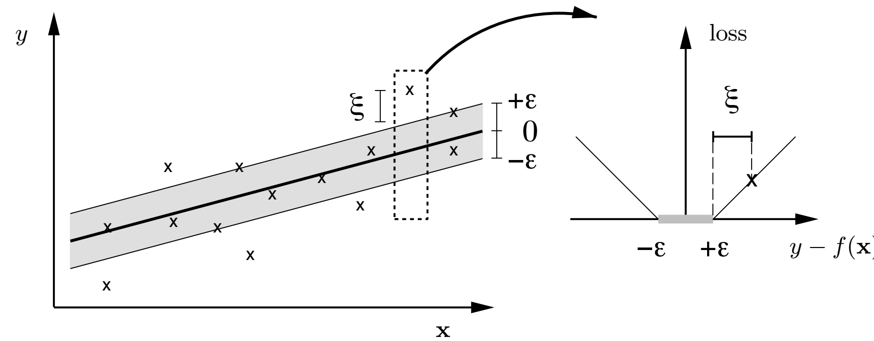

Figure 1.6 In SV regression, a tube with radius ε is fitted to the data. The trade-off between model complexity and points lying outside of the tube (with positive slack variables ξ) is determined by minimizing (1.48).

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

The optimization problem consists of finding

$$\min_{\mathbf{w}\in\mathcal{H},\boldsymbol{\xi}^{(*)}\in\mathbb{R}^m,b\in\mathbb{R}} \quad \tau(\mathbf{w},\boldsymbol{\xi},\boldsymbol{\xi}^*) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^m(\xi_i+\xi_i^*) \tag{1.48}$$

$$\text{subject to} \quad f(\mathbf{x}_i) - y_i \le \varepsilon + \xi_i \tag{1.49}$$

$$y_i - f(\mathbf{x}_i) \le \varepsilon + \xi_i^* \tag{1.50}$$

$$\xi_i, \xi_i^* \ge 0 \tag{1.51}$$

for all $i = 1, \ldots, m$.

Note that according to (1.49) and (1.50), any error smaller than ε does not require a nonzero ξ_i or ξ_i^* and hence does not enter the objective function (1.48).

Generalization to *kernel*-based regression estimation is carried out in complete analogy to the case of pattern recognition. Introducing Lagrange multipliers, one thus arrives at the following optimization problem: for $C > 0, \varepsilon \ge 0$ chosen a priori,

$$\text{maximize } W(\boldsymbol{\alpha},\boldsymbol{\alpha}^*) = -\varepsilon\sum_{i=1}^m(\alpha_i^*+\alpha_i) + \sum_{i=1}^m(\alpha_i^*-\alpha_i)y_i - \frac{1}{2}\sum_{i,j=1}^m(\alpha_i^*-\alpha_i)(\alpha_j^*-\alpha_j)k(x_i,x_j) \tag{1.52}$$

$$\text{subject to} \quad 0 \le \alpha_i, \alpha_i^* \le C, \quad i = 1, \ldots, m, \text{ and } \sum_{i=1}^m(\alpha_i-\alpha_i^*) = 0. \tag{1.53}$$

Regression Function

The regression estimate takes the form

$$f(x) = \sum_{i=1}^{m} (\alpha_i^* - \alpha_i) k(x_i, x) + b, \tag{1.54}$$

where b is computed using the fact that (1.49) becomes an equality with $\xi_i = 0$ if $0 < \alpha_i < C$, and (1.50) becomes an equality with $\xi_i^* = 0$ if $0 < \alpha_i^* < C$ (for details, see Chapter **??**). The solution thus looks quite similar to the pattern recognition case (cf. (1.36) and Figure 1.7).

A number of extensions of this algorithm are possible. From an abstract point of view, we just need some target function which depends on the vector $(\mathbf{w}, \boldsymbol{\xi})$ (cf. (1.48)). 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 [31, 29], cf. Chapter **??**. Moreover, norms other than the 2-norm $\|.\|$ can be used to regularize the solution (see Chapters **??** and **??**).

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

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\left(\nu m \varepsilon + \sum_{i=1}^{m} |y_i - f(\mathbf{x}_i)|_\varepsilon\right) \tag{1.55}$$

instead of (1.47), and treating $\varepsilon \geq 0$ as a parameter that we minimize over. For more details, cf. Chapter **??**.

1.7 Kernel Principal Component Analysis

The kernel method for computing dot products in feature spaces is not restricted to SV machines. 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).

Principal component analysis is perhaps the most common feature extraction algorithm; for details, see Chapter **??**. The term *feature extraction* commonly refers to procedures for extracting (real) numbers from patterns which in some sense represent the crucial information contained in the latter.

PCA in feature space leads to an algorithm called *kernel PCA*, carrying out linear PCA in the feature space $\mathcal{H}$. By the solution of an eigenvalue problem, the algorithm computes nonlinear feature extraction functions

$$f_n(x) = \sum_{i=1}^{m} \alpha_i^n k(x_i, x), \tag{1.56}$$

where, up to a normalization, the α_i^n are the components of the n-th eigenvector of the kernel matrix $K := (k(x_i, x_j))_{ij}$.

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

$$\mathbf{C} := \frac{1}{m} \sum_{i=1}^{m} \Phi(x_i)\Phi(x_i)^\top. \tag{1.57}$$

Here, $\Phi(x_i)^\top$ denotes the the transpose of $\Phi(x_i)$ (see Section **??**).

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

$$\mathbf{C}\mathbf{v} = \lambda\mathbf{v} \tag{1.58}$$

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

$$\mathbf{v} = \sum_{i=1}^{m} \alpha_i \Phi(x_i), \tag{1.59}$$

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

$$m\lambda\boldsymbol{\alpha} = K\boldsymbol{\alpha}, \tag{1.60}$$

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 n-th eigenvector in feature space by

Feature Extraction

$$\langle \mathbf{v}^n, \Phi(x) \rangle = \sum_{i=1}^{m} \alpha_i^n k(x_i, x). \tag{1.61}$$

As in the case of SVMs, the architecture can be visualized by Figure 1.7. Usually, this will be computationally far less expensive than taking the dot product in the feature space explicitly. A toy example is shown in Chapter **??** (Figure **??**).

1.8 Empirical Results and Implementations

Examples of Kernels

Having described the basics of SV machines, 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 noticed that the polynomial kernel

$$k(x, x') = \langle x, x' \rangle^d, \tag{1.62}$$

the Gaussian

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\,\sigma^2}\right), \tag{1.63}$$

and the sigmoid

$$k(x, x') = \tanh\left(\kappa \langle x, x' \rangle + \Theta\right), \tag{1.64}$$

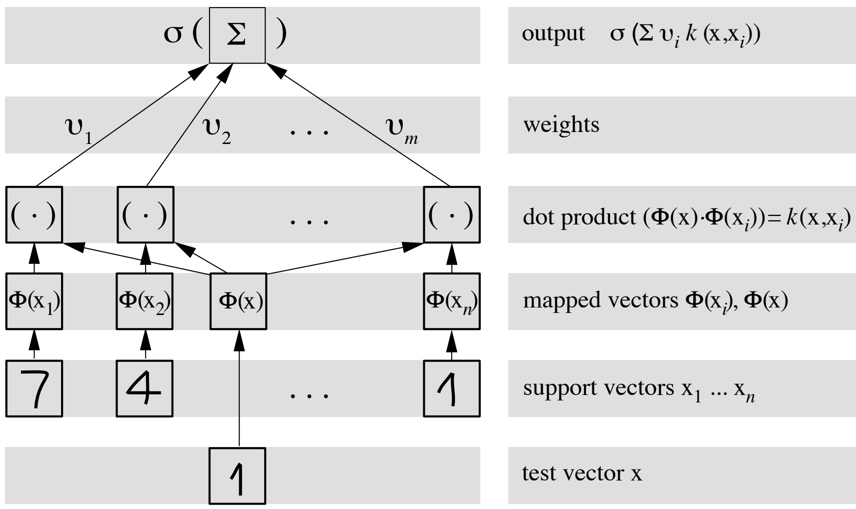

Figure 1.7 Architecture of SV machines and related kernel methods. The input x and the expansion patterns (SVs) x_i (we assume that we are dealing with handwritten digits) are nonlinearly mapped (by Φ) into a feature space $\mathcal{H}$ where dot products are computed. By the use of the kernel k, these two layers are in practice computed in one single step. The results are linearly combined by weights υ_i, found by solving a quadratic program (in pattern recognition, $\upsilon_i = y_i\alpha_i$; in regression estimation, $\upsilon_i = \alpha_i^* - \alpha_i$) 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, $\sigma(x) = x$).

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 (Chapter **??**). 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.

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 SV machines on these issues, leading to the *Virtual SV* method for incorporating prior knowledge about transformation invariances by transforming SVs (Chapter **??**), and the *Reduced Set* method (Chapter **??**) for speeding up classification. This way, SV machines soon became competitive with the best available classifiers on OCR and other object recognition tasks [8], and later even achieved the world record on the main handwritten digit benchmark dataset [11].

Implementation

An initial weakness of SV machines, less apparent in OCR applications which are characterized by low noise levels, was that the size of the quadratic programming problem (Chapter **??**) scaled with the number of support vectors. This was due to the fact that in (1.37), 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 some of the patterns that were initially not identified as SVs turn out to become SVs at a later stage. This procedure is referred to as *chunking*; note that without chunking, the size of the matrix 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 will become nonzero, and all the corresponding examples will become SVs. For this case, decomposition algorithms were proposed [23, 24], 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 [33], MINOS [22], and variants of conjugate gradient descent, such as the optimizers of Bottou [25] and Burges [7]. 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 **??**.

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 [32], and on problems of times series prediction (see [21, 20, 18]). Moreover, the SV method was applied to the solution of inverse function estimation problems ([40]; cf. [37, 42]). For overviews, the interested reader is referred to [7, 27, 30, 10].

References

[1] M. A. Aizerman, É. M. Braverman, and L. I. Rozonoér. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[2] N. Alon, S. Ben-David, N. Cesa-Bianchi, and D. Haussler. Scale–sensitive Dimensions, Uniform Convergence, and Learnability. *Journal of the ACM*, 44(4):615–631, 1997.

[3] M. Anthony and P. Bartlett. *A Theory of Learning in Artificial Neural Networks*. Cambridge University Press, 1999.

[4] P. L. Bartlett and J. Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 43–54, Cambridge, MA, 1999. MIT Press.

[5] C. Berg, J. P. R. Christensen, and P. Ressel. *Harmonic Analysis on Semigroups*. Springer-Verlag, New York, 1984.

[6] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, July 1992. ACM Press.

[7] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[8] C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector learning machines. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 375–381, Cambridge, MA, 1997. MIT Press.

[9] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273 – 297, 1995.

[10] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, UK, 2000.

[11] D. DeCoste and B. Schölkopf. Training invariant support vector machines. *Machine Learning*, 2001. Accepted for publication. Also: Technical Report JPL-MLTR-00-1, Jet Propulsion Laboratory, Pasadena, CA, 2000.

[12] L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Number 31 in Applications of mathematics. Springer, New York, 1996.

[13] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

[14] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, San Diego, 2nd edition, 1990.

[15] H. Goldstein. *Classical Mechanics*. Addison-Wesley, Reading, MA, 1986.

[16] I. Guyon, B. Boser, and V. Vapnik. Automatic capacity tuning of very large VC-dimension classifiers. In Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 147–155. Morgan Kaufmann, San Mateo, CA, 1993.

[17] M. J. Kearns and R. E. Schapire. Efficient distribution-free learning of probabilistic concepts. In *Proc. of the 31st Symposium on the Foundations of Comp. Sci.*, pages 382–391. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[18] D. Mattera and S. Haykin. Support vector machines for dynamic reconstruction of a chaotic system. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 211–242, Cambridge, MA, 1999. MIT Press.

[19] J. Mercer. Functions of positive and negative type and their connection with the theory of

integral equations. *Philos. Trans. Roy. Soc. London*, A 209:415–446, 1909.

[20] S. Mukherjee, E. Osuna, and F. Girosi. Nonlinear prediction of chaotic time series using a support vector machine. In J. Principe, L. Gile, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII — Proceedings of the 1997 IEEE Workshop*, pages 511 – 520, New York, 1997. IEEE.

[21] K.-R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Artificial Neural Networks — ICANN'97*, pages 999 – 1004, Berlin, 1997. Springer Lecture Notes in Computer Science, Vol. 1327.

[22] B. A. Murtagh and M. A. Saunders. MINOS 5.4 user's guide. Technical Report SOL 83.20, Stanford University, 1993.

[23] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In J. Principe, L. Gile, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII — Proceedings of the 1997 IEEE Workshop*, pages 276 – 285, New York, 1997. IEEE.

[24] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA, 1999. MIT Press.

[25] C. Saunders, M. O. Stitson, J. Weston, L. Bottou, B. Schölkopf, and A. Smola. Support vector machine - reference manual. Technical Report CSD-TR-98-03, Department of Computer Science, Royal Holloway, University of London, Egham, UK, 1998. SVM available at http://svm.dcs.rhbnc.ac.uk/.

[26] B. Schölkopf, C. Burges, and V. Vapnik. Incorporating invariances in support vector learning machines. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks — ICANN'96*, pages 47 – 52, Berlin, 1996. Springer Lecture Notes in Computer Science, Vol. 1112.

[27] B. Schölkopf, C. J. C. Burges, and A. J. Smola. *Advances in Kernel Methods — Support Vector Learning*. MIT Press, Cambridge, MA, 1999.

[28] B. Schölkopf, A. Smola, R. C. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12:1207 – 1245, 2000.

[29] A. Smola, B. Schölkopf, and K.-R. Müller. The connection between regularization operators and support vector kernels. *Neural Networks*, 11:637–649, 1998.

[30] A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans. *Advances in Large Margin Classifiers*. MIT Press, Cambridge, MA, 2000.

[31] A. J. Smola and B. Schölkopf. On a kernel–based method for pattern recognition, regression, approximation and operator inversion. *Algorithmica*, 22:211–231, 1998.

[32] M. Stitson, A. Gammerman, V. Vapnik, V. Vovk, C. Watkins, and J. Weston. Support vector regression with ANOVA decomposition kernels. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 285–292, Cambridge, MA, 1999. MIT Press.

[33] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, Hingham, MA, 1997.

[34] V. Vapnik. *Estimation of Dependences Based on Empirical Data [in Russian]*. Nauka, Moscow, 1979. (English translation: Springer Verlag, New York, 1982).

[35] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, NY, 1995.

[36] V. Vapnik. *Statistical Learning Theory*. Wiley, NY, 1998.

[37] V. Vapnik. Three remarks on the support vector method of function estimation. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 25–42, Cambridge, MA, 1999. MIT Press.

[38] V. Vapnik and A. Chervonenkis. A note on one class of perceptrons. *Automation and Remote Control*, 25, 1964.

[39] V. Vapnik and A. Chervonenkis. *Theory of Pattern Recognition [in Russian]*. Nauka, Moscow, 1974. (German Translation: W. Wapnik & A. Tscherwonenkis, *Theorie der Zeichenerkennung*, Akademie–Verlag, Berlin, 1979).

[40] V. Vapnik, S. Golowich, and A. Smola. Support vector method for function approximation, regression estimation, and signal processing. In M. Mozer, M. Jordan, and T. Petsche, editors,

Advances in Neural Information Processing Systems 9, pages 281–287, Cambridge, MA, 1997. MIT Press.

[41] V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24, 1963.

[42] J. Weston. Leave–one–out support vector machines. In *Proceedings of the International Joint Conference on Artifical Intelligence*, Sweden, 1999.

[43] R. C. Williamson, A. J. Smola, and B. Schölkopf. Generalization performance of regularization networks and support vector machines via entropy numbers of compact operators. Technical Report 19, NeuroCOLT, http://www.neurocolt.com, 1998. Accepted for publication in IEEE Transactions on Information Theory.

Index