2 Kernel Classifiers from a Machine Learning Perspective

This chapter presents the machine learning approach to learning kernel classifiers. After a short introduction to the problem of learning a linear classifier, it shows how learning can be viewed as an optimization task. As an example, the classical perceptron algorithm is presented. This algorithm is an implementation of a more general principle known as *empirical risk minimization*. The chapter also presents a descendant of this principle, known as *regularized (structural) risk minimization*. Both these principles can be applied in the primal or dual space of variables. It is shown that the latter is computationally less demanding if the method is extended to nonlinear classifiers in input space. Here, the *kernel technique* is the essential method used to invoke the nonlinearity in input space. The chapter presents several families of kernels that allow linear classification methods to be applicable even if no vectorial representation is given, e.g., strings. Following this, the *support vector* method for classification learning is introduced. This method elegantly combines the kernel technique and the principle of structural risk minimization. The chapter finishes with a presentation of a more recent kernel algorithm called *adaptive margin machines*. In contrast to the support vector method, the latter aims at minimizing a leave-one-out error bound rather than a structural risk.

2.1 The Basic Setting

The task of classification learning is the problem of finding a good strategy to assign class labels to objects based on past observations of object-class pairs. We shall only assume that all objects x are contained in the set $\mathcal{X}$, often referred to as the *input space*. Let $\mathcal{Y}$ be a finite set of classes called the *output space*. If not otherwise stated, we will only consider the two-element output space $\{-1, +1\}$.

in which case the learning problem is called a *binary classification* learning task. Suppose we are given a sample of m training objects,

$$x = (x_1, \ldots, x_m) \in \mathcal{X}^m ,$$

together with a sample of corresponding class labels,

$$y = (y_1, \ldots, y_m) \in \mathcal{Y}^m .$$

We will often consider the labeled training sample,[1]

$$z = (x, y) = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m = \mathcal{Z}^m ,$$

and assume that z is a sample drawn identically and independently distributed (iid) according to some unknown probability measure $\mathbf{P}_Z$.

Definition 2.1 (Learning problem) *The* learning problem *is to find the unknown (functional) relationship $h \in \mathcal{Y}^{\mathcal{X}}$ between objects $x \in \mathcal{X}$ and targets $y \in \mathcal{Y}$ based solely on a sample $z = (x, y) = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$ of size $m \in \mathbb{N}$ drawn iid from an unknown distribution $\mathbf{P}_{XY}$. If the output space $\mathcal{Y}$ contains a finite number $|\mathcal{Y}|$ of elements then the task is called a* classification learning problem.

Of course, having knowledge of $\mathbf{P}_{XY} = \mathbf{P}_Z$ is sufficient for identifying this relationship as for all objects x,

$$\mathbf{P}_{Y|X=x}(y) = \frac{\mathbf{P}_Z((x, y))}{\mathbf{P}_X(x)} = \frac{\mathbf{P}_Z((x, y))}{\sum_{\tilde{y} \in \mathcal{Y}} \mathbf{P}_Z((x, \tilde{y}))} . \tag{2.1}$$

Thus, for a given object $x \in \mathcal{X}$ we could evaluate the distribution $\mathbf{P}_{Y|X=x}$ over class labels and decide on the class label $\hat{y} \in \mathcal{Y}$ with the largest probability $\mathbf{P}_{Y|X=x}(\hat{y})$. Estimating $\mathbf{P}_Z$ based on the given sample z, however, poses a nontrivial problem. In the (unconstrained) class of all probability measures, the *empirical measure*

$$\nu_z((x, y)) = \frac{|\{i \in \{1, \ldots, m\} \mid z_i = (x, y)\}|}{m} \tag{2.2}$$

1 Though mathematically the training sample is a *sequence* of iid drawn object-class pairs (x, y) we sometimes take the liberty of calling the training sample a *training set*. The notation $z \in z$ then refers to the fact that there exists an element z_i in the sequence z such that $z_i = z$.

is among the "most plausible" ones, because

$$\mathbf{v}_{\mathbf{z}}(\{z_1, \ldots, z_m\}) = \sum_{i=1}^{m} \mathbf{v}_{\mathbf{z}}(z_i) = 1 .$$

However, the corresponding "identified" relationship $h_{\mathbf{v}_{\mathbf{z}}} \in \mathcal{Y}^{\mathcal{X}}$ is unsatisfactory because

$$h_{\mathbf{v}_{\mathbf{z}}}(x) = \sum_{x_i \in x} y_i \cdot \mathbf{I}_{x=x_i}$$

assigns zero probability to all unseen objects-class pairs and thus cannot be used for predicting a class label given a new object $x \in \mathcal{X}$. In order to resolve this difficulty, we need to constrain the set $\mathcal{Y}^{\mathcal{X}}$ of possible mappings from objects $x \in \mathcal{X}$ to class labels $y \in \mathcal{Y}$. Often, such a restriction is imposed by assuming a given *hypothesis space* $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$ of functions[2] $h : \mathcal{X} \to \mathcal{Y}$. Intuitively, similar objects x_i should be mapped to the same class y_i. This is a very reasonable assumption if we wish to infer class labels on unseen objects x *based on a given training sample* $\mathbf{z}$ *only*.

A convenient way to model *similarity* between objects is through an inner product function $\langle \cdot, \cdot \rangle$ which has the appealing property that its value is maximal whenever its arguments are equal. In order to employ inner products to measure similarity between objects we need to represent them in an inner product space which we assume to be ℓ_2^n (see Definition A.39).

Definition 2.2 (Features and feature space) *A function $\phi_i : \mathcal{X} \to \mathbb{R}$ that maps each object $x \in \mathcal{X}$ to a real value $\phi_i(x)$ is called a* feature. *Combining n features $\phi_1, \ldots, \phi_n$ results in a* feature mapping $\boldsymbol{\phi} : \mathcal{X} \to \mathcal{K} \subseteq \ell_2^n$ *and the space $\mathcal{K}$ is called a* feature space.

In order to avoid an unnecessarily complicated notation we will abbreviate $\boldsymbol{\phi}(x)$ by $\mathbf{x}$ for the rest of the book. The vector $\mathbf{x} \in \mathcal{K}$ is also called the *representation* of $x \in \mathcal{X}$. This should not be confused with the training sequence $\boldsymbol{x}$ which results in an $m \times n$ matrix $\mathbf{X} = (\mathbf{x}_1'; \ldots; \mathbf{x}_m')$ when applying $\boldsymbol{\phi}$ to it.

Example 2.3 (Handwritten digit recognition) *The important task of classifying handwritten digits is one of the most prominent examples of the application of learning algorithms. Suppose we want to automatically construct a procedure*

2 Since each h is a hypothetical mapping to class labels, we synonymously use *classifier*, *hypothesis* and *function* to refer to h.

which can assign digital images to the classes "image is a picture of 1" and "image is not a picture of 1". Typically, each feature $\phi_l : \mathcal{X} \to \mathbb{R}$ is the intensity of ink at a fixed picture element, or pixel, *of the image. Hence, after digitalization at $N \times N$ pixel positions, we can represent each image as a high dimensional vector $\mathbf{x}$ (to be precise, N^2–dimensional). Obviously, only a small subset of the N^2–dimensional space is occupied by handwritten digits*[3]*, and, due to noise in the digitization, we might have the same picture x mapped to different vectors $\mathbf{x}_i$, $\mathbf{x}_j$. This is assumed encapsulated in the probability measure $\mathbf{P}_{\mathbf{X}}$. Moreover, for small N, similar pictures $x_i \approx x_j$ are mapped to the same data vector $\mathbf{x}$ because the single pixel positions are too coarse a representation of a single image. Thus, it seems reasonable to assume that one could hardly find a deterministic mapping from N^2–dimensional vectors to the class "picture of 1". This gives rise to a probability measure $\mathbf{P}_{Y|\mathbf{X}=\mathbf{x}}$. Both these uncertainties—which in fact constitute the basis of the learning problem—are expressed via the unknown probability measure $\mathbf{P}_Z$ (see equation (2.1)).*

In this book, we will be concerned with linear functions or classifiers only. Let us formally define what we mean when speaking about linear classifiers.

Definition 2.4 (Linear function and linear classifier) *Given a feature mapping $\boldsymbol{\phi} : \mathcal{X} \to \mathcal{K} \subseteq \ell_2^n$, the function $f : \mathcal{X} \to \mathbb{R}$ of the form*[4]

$$f_{\mathbf{w}}(x) = \langle \boldsymbol{\phi}(x), \mathbf{w} \rangle = \langle \mathbf{x}, \mathbf{w} \rangle$$

is called a linear function *and the n–dimensional vector $\mathbf{w} \in \mathcal{K}$ is called a* weight vector. *A* linear classifier *is obtained by thresholding a linear function,*

$$h_{\mathbf{w}}(x) = \operatorname{sign}(\langle \mathbf{x}, \mathbf{w} \rangle) . \tag{2.3}$$

Clearly, the intuition that similar objects are mapped to similar class labels is satisfied by such a model because, by the Cauchy-Schwarz inequality (see Theorem A.106), we know that

$$\left| \langle \mathbf{w}, \mathbf{x}_i \rangle - \langle \mathbf{w}, \mathbf{x}_j \rangle \right| = \left| \langle \mathbf{w}, \mathbf{x}_i - \mathbf{x}_j \rangle \right| \leq \|\mathbf{w}\| \cdot \|\mathbf{x}_i - \mathbf{x}_j\| ;$$

3 To see this, imagine that we generate an image by tossing a coin N^2 times and mark a black dot in a $N \times N$ array, if the coin shows head. Then, it is very unlikely that we will obtain an image of a digit. This outcome is expected as digits presumably have a pictorial structure in common.

4 In order to highlight the dependence of f on $\mathbf{w}$, we use $f_{\mathbf{w}}$ when necessary.

that is, whenever two data points are close in feature space (small $\|\mathbf{x}_i - \mathbf{x}_j\|$), their difference in the real-valued output of a hypothesis with weight vector $\mathbf{w} \in \mathcal{K}$ is also small. It is important to note that the classification $h_{\mathbf{w}}(x)$ remains unaffected if we rescale the weight $\mathbf{w}$ by some positive constant,

$$\forall \lambda > 0 : \forall x \in \mathcal{X} : \quad \operatorname{sign}(\langle \mathbf{x}, \lambda \mathbf{w} \rangle) = \operatorname{sign}(\lambda \langle \mathbf{x}, \mathbf{w} \rangle) = \operatorname{sign}(\langle \mathbf{x}, \mathbf{w} \rangle) . \tag{2.4}$$

Thus, if not stated otherwise, we assume the weight vector $\mathbf{w}$ to be of unit length,

$$\mathcal{F} = \{x \mapsto \langle \mathbf{x}, \mathbf{w} \rangle \mid \mathbf{w} \in \mathcal{W}\} \subseteq \mathbb{R}^{\mathcal{X}} , \tag{2.5}$$

$$\mathcal{W} = \{\mathbf{w} \in \mathcal{K} \mid \|\mathbf{w}\| = 1\} \subset \mathcal{K} , \tag{2.6}$$

$$\mathcal{H} = \left\{ h_{\mathbf{w}} \stackrel{\text{def}}{=} \operatorname{sign}(f_{\mathbf{w}}) \mid f_{\mathbf{w}} \in \mathcal{F} \right\} \subseteq \mathcal{Y}^{\mathcal{X}} . \tag{2.7}$$

Ergo, the set $\mathcal{F}$, also referred to as the *hypothesis space*, is isomorphic to the unit hypersphere $\mathcal{W}$ in $\mathbb{R}^n$ (see Figure 2.1).

The task of learning reduces to finding the "best" classifier f^* in the hypothesis space $\mathcal{F}$. The most difficult question at this point is: "How can we measure the goodness of a classifier f? We would like the goodness of a classifier to be

- strongly dependent on the unknown measure $\mathbf{P}_Z$; otherwise, we would not have a learning problem because f^* could be determined without knowledge of the underlying relationship between objects and classes expressed via $\mathbf{P}_Z$.
- pointwise w.r.t. the object-class pairs (x, y) due to the independence assumption made for z.
- a positive, real-valued function, making the maximization task computationally easier.

All these requirements can be encapsulated in a fixed *loss function* $l : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$. Here $l(f(x), y)$ measures how costly it is when the prediction at the data point x is $f(x)$ but the true class is y. It is natural to assume that $l(+\infty, +1) = l(-\infty, -1) = 0$, that is, the greater $y \cdot f(x)$ the better the prediction of $f(x)$ was. Based on the loss l it is assumed that the goodness of f is the expected loss $\mathbf{E}_{XY}\left[l(f(\mathsf{X}), \mathsf{Y})\right]$, sometimes referred to as the expected risk. In summary, the ultimate goal of learning can be described as:

> Based on the training sample $z \in \mathcal{Z}^m$, a hypothesis space $\mathcal{F} \subseteq \mathbb{R}^{\mathcal{X}}$ and a loss function $l : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$ find the function
>
> $$f^* = \operatorname*{argmin}_{f \in \mathcal{F}} \mathbf{E}_{XY}\left[l(f(\mathsf{X}), \mathsf{Y})\right] .$$

Assuming an unknown, but fixed, measure P_Z over the object-class space $\mathcal{Z}$ we can view the expectation value $\mathsf{E}_{XY}\left[l\left(f\left(X\right), Y\right)\right]$ of the loss as an *expected risk functional* over $\mathcal{F}$.

Definition 2.5 (Expected risk) *Given a loss $l : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$ and a measure P_{XY}, the functional*

$$R\left[f\right] \stackrel{\text{def}}{=} \mathsf{E}_{XY}\left[l\left(f\left(X\right), Y\right)\right], \tag{2.8}$$

is called expected risk *or* expected loss *of a function $f \in \mathcal{F} \subseteq \mathbb{R}^{\mathcal{X}}$, respectively. If the loss function $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ maps from the predicted and true class labels to the reals, the expected risk is also defined by (2.8) but this time w.r.t. $h \in \mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$.*

Example 2.6 (Classification loss) *In the case of classification learning, a natural measure of goodness of a classifier $h \in \mathcal{H}$ is the probability of assigning a new object to the wrong class, i.e., $\mathsf{P}_{XY}\left(h\left(X\right) \neq Y\right)$. In order to cast this into a loss-based framework we exploit the basic fact that $\mathsf{P}\left(A\right) = \mathsf{E}\left[\mathsf{I}_A\right]$ for some A. As a consequence, using the* zero-one loss *$l_{0-1} : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$ for real-valued functions*

$$l_{0-1}\left(f\left(x\right), y\right) \stackrel{\text{def}}{=} \mathsf{I}_{yf(x) \leq 0}, \tag{2.9}$$

renders the task of finding the classifier with minimal misclassification probability as a risk minimization task. Note that, due to the fact that $y \in \{-1, +1\}$, the zero-one loss in equation (2.9) is a special case of the more general loss function $l_{0-1} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$

$$l_{0-1}\left(h\left(x\right), y\right) \stackrel{\text{def}}{=} \mathsf{I}_{h(x) \neq y}. \tag{2.10}$$

Example 2.7 (Cost matrices) *Returning to Example 2.3 we see that the loss given by equation (2.9) is inappropriate for the task at hand. This is due to the fact that there are approximately ten times more "no pictures of 1" than "pictures of 1". Therefore, a classifier assigning each image to the class "no picture of 1" (this classifier is also known as the* default classifier*) would have an expected risk of about 10%. In contrast, a classifier assigning each image to the class "picture of 1" would have an expected risk of about 90%. To correct this imbalance of prior probabilities $\mathsf{P}_Y\left(+1\right)$ and $\mathsf{P}_Y\left(-1\right)$ one could define a 2×2 cost matrix*

$$\mathbf{C} = \begin{pmatrix} 0 & c_{12} \\ c_{21} & 0 \end{pmatrix}.$$

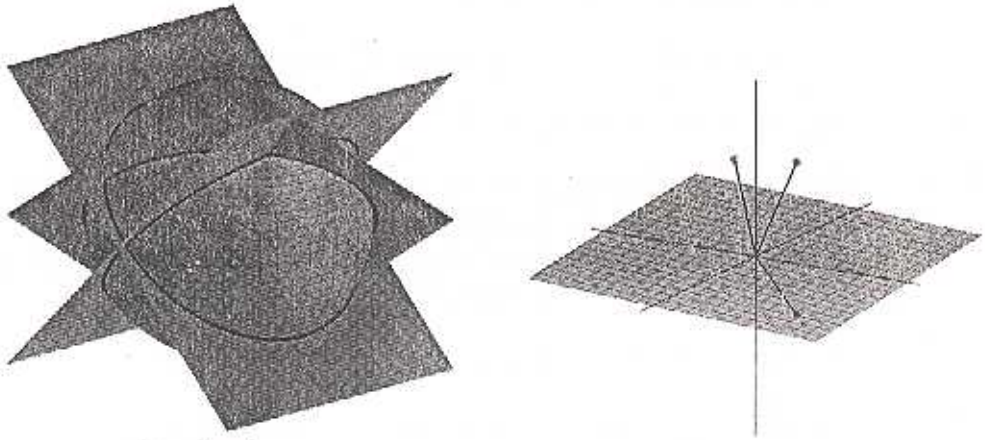

Figure 2.1 **(Left)** The hypothesis space $\mathcal{W}$ for linear classifiers in $\mathbb{R}^3$. Each single point x defines a plane in $\mathbb{R}^3$ and thus incurs a grand circle $\{\mathbf{w} \in \mathcal{W} \mid \langle \mathbf{x}, \mathbf{w} \rangle = 0\}$ in hypothesis space (black lines). The three data points in the right picture induce the three planes in the left picture. **(Right)** Considering a fixed classifier $\mathbf{w}$ (single dot on the left) the decision plane $\{\mathbf{x} \in \mathbb{R}^3 \mid \langle \mathbf{x}, \mathbf{w} \rangle = 0\}$ is shown.

Let $\mathbf{1}_y$ *and* $\mathbf{1}_{\text{sign}(f(x))}$ *denote the* 2×1 *indicator vectors of the true class and the classification made by* $f \in \mathcal{F}$ *at* $x \in \mathcal{X}$. *Then we have a* cost matrix classification loss l_C *by*

$$l_C(f(x), y) \stackrel{\text{def}}{=} \mathbf{1}_y' \mathbf{C} \mathbf{1}_{\text{sign}(f(x))} = \begin{cases} c_{12} & y = +1 \text{ and } f(x) < 0 \\ c_{21} & y = -1 \text{ and } f(x) > 0 \\ 0 & \text{otherwise}. \end{cases}$$

Obviously, setting $c_{12} = \mathbf{P}_Y(-1)$ *and* $c_{21} = \mathbf{P}_Y(+1)$ *leads to equal risks for both default classifiers and thus allows the incorporation of prior knowledge on the probabilities* $\mathbf{P}_Y(+1)$ *and* $\mathbf{P}_Y(-1)$.

Remark 2.8 (Geometrical picture) *Linear classifiers, parameterized by a weight vector* $\mathbf{w}$, *are hyperplanes passing through the origin in feature space* $\mathcal{K}$. *Each classifier divides the feature space into two open half spaces,* $X_{+1}(\mathbf{w}) \subset \mathcal{K}$, $X_{-1}(\mathbf{w}) \subset$

$\mathcal{K}$ by the hyperplane[5] $X_0(\mathbf{w}) \subset \mathcal{K}$ *using the following rule,*

$$X_y(\mathbf{w}) = \{\mathbf{x} \in \mathcal{K} \mid \operatorname{sign}(\langle \mathbf{x}, \mathbf{w} \rangle) = y\} .$$

Considering the images of $X_0(\mathbf{w})$ *in object space* $\mathcal{X}$

$$\widetilde{X}_0(\mathbf{w}) = \{x \in \mathcal{X} \mid \langle \mathbf{x}, \mathbf{w} \rangle = 0\} ,$$

this set is sometimes called the decision surface. *Our hypothesis space* $\mathcal{W}$ *for weight vectors* $\mathbf{w}$ *is the unit hypersphere in* $\mathbb{R}^n$ *(see equation (2.6)). Hence, having fixed* $\mathbf{x}$*, the unit hypersphere* $\mathcal{W}$ *is subdivided into three disjoint sets* $W_{+1}(\mathbf{x}) \subset \mathcal{W}$, $W_{-1}(\mathbf{x}) \subset \mathcal{W}$ *and* $W_0(\mathbf{x}) \subset \mathcal{W}$ *by exactly the same rule, i.e.,*

$$W_y(\mathbf{x}) = \{\mathbf{w} \in \mathcal{W} \mid \operatorname{sign}(\langle \mathbf{x}, \mathbf{w} \rangle) = y\} .$$

As can be seen in Figure 2.1 (left), for a finite sample $\boldsymbol{x} = (x_1, \ldots, x_m)$ *of training objects and any vector* $\boldsymbol{y} = (y_1, \ldots, y_m) \in \{-1, +1\}^m$ *of labelings the resulting* equivalence classes

$$W_z = \bigcap_{i=1}^{m} W_{y_i}(\mathbf{x}_i)$$

are (open) convex polyhedra. Clearly, the labeling of the x_i *determines the training error of each equivalence class*

$$W_z = \{\mathbf{w} \in \mathcal{W} \mid \forall i \in \{1, \ldots, m\} : \operatorname{sign}(\langle \mathbf{x}_i, \mathbf{w} \rangle) = y_i\} .$$

2.2 Learning by Risk Minimization

Apart from algorithmical problems, as soon as we have a fixed object space $\mathcal{X}$, a fixed set (or space) $\mathcal{F}$ of hypotheses and a fixed loss function l, learning reduces to a pure optimization task on the functional $R[f]$.

Definition 2.9 (Learning algorithm) *Given an object space* $\mathcal{X}$*, an output space* $\mathcal{Y}$ *and a fixed set* $\mathcal{F} \subseteq \mathbb{R}^{\mathcal{X}}$ *of functions mapping* $\mathcal{X}$ *to* $\mathbb{R}$*, a* learning algorithm $\mathcal{A}$

5 With a slight abuse of notation, we use $\operatorname{sign}(0) = 0$.

for the hypothesis space $\mathcal{F}$ *is a mapping*[6]

$$\mathcal{A} : \bigcup_{m=1}^{\infty} (\mathcal{X} \times \mathcal{Y})^m \to \mathcal{F} .$$

The biggest difficulty so far is that we have no knowledge of the function to be optimized, i.e., we are only given an iid sample z instead of the full measure $\mathbf{P}_Z$. Thus, it is impossible to solve the learning problem exactly. Nevertheless, for any learning method we shall require its performance to improve with increasing training sample size, i.e., the probability of drawing a training sample z such that the generalization error is large will decrease with increasing m. Here, the generalization error is defined as follows.

Definition 2.10 (Generalization error) *Given a learning algorithm $\mathcal{A}$ and a loss $l : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$ the* generalization error of $\mathcal{A}$ *is defined as*

$$\mathcal{R}[\mathcal{A}, z] \stackrel{\text{def}}{=} R[\mathcal{A}(z)] - \inf_{f \in \mathcal{F}} R[f] .$$

In other words, the generalization error measures the deviation of the expected risk of the function learned from the minimum expected risk.

The most well known learning principle is the *empirical risk minimization* (ERM) principle. Here, we replace $\mathbf{P}_Z$ by $\mathbf{v}_z$, which contains all knowledge that can be drawn from the training sample z. As a consequence the expected risk becomes an empirically computable quantity known as the empirical risk.

Definition 2.11 (Empirical risk) *Given a training sample $z \in (\mathcal{X} \times \mathcal{Y})^m$ the functional*

$$R_{\text{emp}}[f, z] \stackrel{\text{def}}{=} \frac{1}{m} \sum_{i=1}^{m} l(f(x_i), y_i) , \tag{2.11}$$

is called the empirical risk functional *over $f \in \mathcal{F} \subseteq \mathbb{R}^{\mathcal{X}}$ or* training error *of f, respectively.*

6 The definition for the case of hypotheses $h \in \mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$ is equivalent.

By construction, R_{emp} can be minimized solely on the basis of the training sample z. We can write any ERM algorithm in the form,

$$\mathcal{A}_{\text{ERM}}(z) \stackrel{\text{def}}{=} \underset{f \in \mathcal{F}}{\operatorname{argmin}} \; R_{\text{emp}}[f, z] \,. \tag{2.12}$$

In order to be a consistent learning principle, the expected risk $R[\mathcal{A}_{\text{ERM}}(z)]$ must converge to the minimum expected risk $R[f^*]$, i.e.,

$$\forall \varepsilon > 0: \qquad \lim_{m \to \infty} \mathbf{P}_{Z^m} \left(R\left[\mathcal{A}_{\text{ERM}}(\mathbf{Z})\right] - R\left[f^*\right] > \varepsilon \right) = 0 , \tag{2.13}$$

where the randomness is due to the random choice of the training sample z.

It is known that the empirical risk $R_{\text{emp}}[f, z]$ of a *fixed* function f converges toward $R[f]$ at an exponential rate w.r.t. m for any probability measure $\mathbf{P}_Z$ (see Subsection A.5.2). Nonetheless, it is not clear whether this holds when we consider the empirical risk minimizer $\mathcal{A}_{\text{ERM}}(z)$ given by equation (2.12) because this function changes over the random choice of training samples z. We shall see in Chapter 4 that the finiteness of the number n of feature space dimensions completely determines the consistency of the ERM principle.

2.2.1 The (Primal) Perceptron Algorithm

The first iterative procedure for learning linear classifiers presented is the *perceptron learning algorithm* proposed by F. Rosenblatt. The learning algorithm is given on page 323 and operates as follows:

1. At the start the weight vector $\mathbf{w}$ is set to $\mathbf{0}$.

2. For each training example (x_i, y_i) it is checked whether the current hypothesis correctly classifies or not. This can be achieved by evaluating the sign of $y_i \langle \mathbf{x}_i, \mathbf{w} \rangle$. If the ith training sample is not correctly classified then the misclassified pattern $\mathbf{x}_i$ is added to or subtracted from the current weight vector depending on the correct class label y_i. In summary, the weight vector $\mathbf{w}$ is updated to $\mathbf{w} + y_i \mathbf{x}_i$.

3. If no mistakes occur during an iteration through the training sample z the algorithm stops and outputs $\mathbf{w}$.

The optimization algorithm is a mistake-driven procedure, and it assumes the existence of a version space $V(z) \subseteq \mathcal{W}$, i.e., it assumes that there exists at least one classifier f such that $R_{\text{emp}}[f, z] = 0$.

Definition 2.12 (Version space) *Given the training sample* $z = (x, y) \in (\mathcal{X} \times \mathcal{Y})^m$ *and a hypothesis space* $\mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$, *we call*

$$V_{\mathcal{H}}(z) \stackrel{\text{def}}{=} \{h \in \mathcal{H} \mid \forall i \in \{1, \ldots, m\} : h(x_i) = y_i\} \subseteq \mathcal{H}$$

the version space, *i.e., the set of all classifiers consistent with the training sample. In particular, for linear classifiers given by (2.5)–(2.7) we synonymously call the set of consistent weight vectors*

$$V(z) \stackrel{\text{def}}{=} \{\mathbf{w} \in \mathcal{W} \mid \forall i \in \{1, \ldots, m\} : y_i \langle \mathbf{x}_i, \mathbf{w} \rangle > 0\} \subseteq \mathcal{W}$$

the version space.

Since our classifiers are linear in feature space, such training samples are called *linearly separable*. In order that the perceptron learning algorithm works for *any* training sample it must be ensured that the unknown probability measure $\mathbf{P}_Z$ satisfies $R[f^*] = 0$. Viewed differently, this means that $\mathbf{P}_{Y|X=x}(y) = \mathbf{I}_{y=h^*(x)}$, $h^* \in \mathcal{H}$, where h^* is sometimes known as the *teacher perceptron*. It should be noticed that the number of parameters learned by the perceptron algorithm is n, i.e., the dimensionality of the feature space $\mathcal{K}$. We shall call this space of parameters the *primal space*, and the corresponding algorithm the *primal perceptron learning algorithm*. As depicted in Figure 2.2, perceptron learning is best viewed as starting from an arbitrary[7] point $\mathbf{w}_0$ on the hypersphere $\mathcal{W}$, and each time we observe a misclassification with a training example (x_i, y_i), we update $\mathbf{w}_t$ toward the misclassified training object $y_i \mathbf{x}_i$ (see also Figure 2.1 (left)). Thus, geometrically, the perceptron learning algorithm performs a walk through the primal parameter space with each step made in the direction of decreasing training error. Note, however, that in the formulation of the algorithm given on page 323 we do not normalize the weight vector $\mathbf{w}$ after each update.

2.2.2 Regularized Risk Functionals

One possible method of overcoming the lack of knowledge about $\mathbf{P}_Z$ is to replace it by its empirical estimate $\mathbf{v}_z$. This principle, discussed in the previous section, justifies the perceptron learning algorithm. However, minimizing the empirical risk, as done by the perceptron learning algorithm, has several drawbacks:

7 Although in algorithm 1 on page 323 we start at $\mathbf{w}_0 = \mathbf{0}$ it is not necessary to do so.

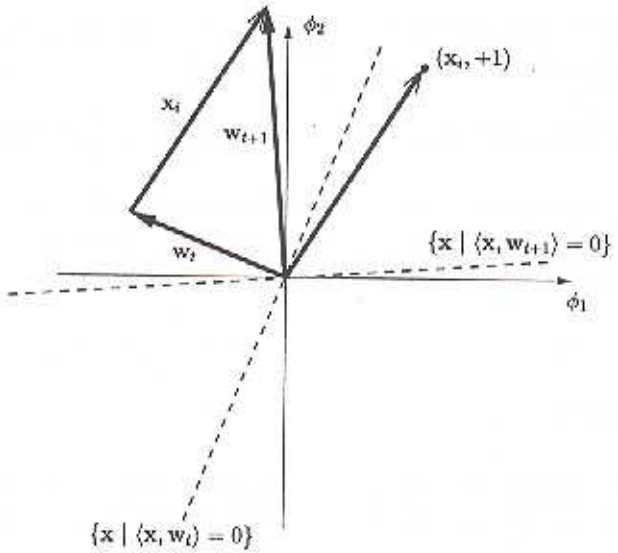

Figure 2.2 A geometrical picture of the update step in the perceptron learning algorithm in $\mathbb{R}^2$. Evidently, $\mathbf{x}_i \in \mathbb{R}^2$ is misclassified by the linear classifier (dashed line) having normal $\mathbf{w}_t$ (solid line with arrow). Then, the update step amounts to changing $\mathbf{w}_t$ into $\mathbf{w}_{t+1} = \mathbf{w}_t + y_i \mathbf{x}_i$ and thus $y_i \mathbf{x}_i$ "attracts" the hyperplane. After this step, the misclassified point $\mathbf{x}_i$ is correctly classified.

1. Many examples are required to ensure a small generalization error $R[\mathcal{A}_{\mathrm{ERM}}, z]$ with high probability taken over the random choice of z.

2. There is no unique minimum, i.e., each weight vector $\mathbf{w} \in V(z)$ in version space parameterizes a classifier $f_{\mathbf{w}}$ that has $R_{\mathrm{emp}}[f_{\mathbf{w}}, z] = 0$.

3. Without any further assumptions on $\mathbf{P}_Z$ the number of steps until convergence of the perceptron learning algorithm is not bounded.

- A training sample $z \in \mathcal{Z}^m$ that is linearly separable in feature space is required.

The second point in particular shows that ERM learning makes the learning task an *ill-posed* one (see Appendix A.4): A slight variation $\tilde{z}$ in the training sample z might lead to a large deviation between the expected risks of the classifiers learned using the ERM principle, $\left| R[\mathcal{A}_{\mathrm{ERM}}(z)] - R\left[\mathcal{A}_{\mathrm{ERM}}(\tilde{z})\right] \right|$. As will be seen in Part II of this book, a very influential factor in this deviation is the possibility of the hypothesis space $\mathcal{F}$ adopting different labelings y for randomly drawn objects x. The more diverse the set of functions a hypothesis space contains, the more easily

it can produce a given labeling y regardless of how bad the subsequent prediction might be on new, as yet unseen, data points $z = (x, y)$. This effect is also known as *overfitting*, i.e., the empirical risk as given by equation (2.11) is much smaller than the expected risk (2.8) we originally aimed at minimizing.

One way to overcome this problem is the method of *regularization*. In our example this amounts to introducing a regularizer a-priori, that is, a functional $\Omega : \mathcal{F} \to \mathbb{R}^+$, and defining the solution to the learning problem to be

$$\mathcal{A}_\Omega(z) \stackrel{\text{def}}{=} \operatorname*{argmin}_{f \in \mathcal{F}} \underbrace{R_{\text{emp}}[f, z] + \lambda \Omega[f]}_{R_{\text{reg}}[f, z]} . \tag{2.14}$$

The idea of regularization is to restrict the space of solutions to compact subsets of the (originally overly large) space $\mathcal{F}$. This can be achieved by requiring the set $F_\varepsilon = \{f \mid \Omega[f] \leq \varepsilon\} \subseteq \mathcal{F}$ to be compact for each positive number $\varepsilon > 0$. This, in fact, is the essential requirement for any regularizer Ω. Then, if we decrease λ for increasing training sample sizes in the right way, it can be shown that the regularization method leads to f^* as $m \to \infty$ (see equation (2.13)). Clearly, $0 \leq \lambda < \infty$ controls the amount of regularization. Setting $\lambda = 0$ is equivalent to minimizing only the empirical risk. In the other extreme, considering $\lambda \to \infty$ amounts to discounting the sample and returning the classifier which minimizes Ω alone. The regularizer Ω can be thought of as a penalization term for the "complexity" of particular classifiers.

Another view of the regularization method can be obtained from the statistical study of learning algorithms. This will be discussed in greater detail in Part II of this book but we shall put forward the main idea here. We shall see that there exist several measures of "complexity" of hypothesis spaces, the VC dimension being the most prominent thereof. V. Vapnik suggested a learning principle which he called *structural risk minimization* (SRM). The idea behind SRM is to, a-priori, define a structuring of the hypothesis space $\mathcal{F}$ into nested subsets $\mathcal{F}_0 \subset \mathcal{F}_1 \subset \cdots \subseteq \mathcal{F}$ of increasing complexity. Then, in each of the hypothesis spaces $\mathcal{F}_i$ empirical risk minimization is performed. Based on results from statistical learning theory, an SRM algorithm returns the classifier with the smallest *guaranteed risk*[8]. This can be related to the algorithm (2.14), if $\Omega[f]$ is the complexity value of f given by the used bound for the guaranteed risk.

From a Bayesian perspective, however, the method of regularization is closely related to *maximum-a-posteriori* (MAP) estimation. To see this, it suffices to

8 This is a misnomer as it refers to the value of an upper bound at a fixed confidence level and can in no way be guaranteed.

express the empirical risk as the negative log-probability of the training sample z, given a classifier f. In general, this can be achieved by

$$\begin{aligned} \mathsf{P}_{Z^m|\mathsf{F}=f}(z) &= \prod_{i=1}^{m} \mathsf{P}_{\mathsf{Y}|\mathsf{X}=x_i,\mathsf{F}=f}(y_i)\, \mathsf{P}_{\mathsf{X}|\mathsf{F}=f}(x_i)\,, \\ \mathsf{P}_{\mathsf{Y}|\mathsf{X}=x,\mathsf{F}=f}(y) &= \frac{\exp\left(-l\left(f(x), y\right)\right)}{\sum_{\tilde{y}\in\mathcal{Y}} \exp\left(-l\left(f(x), \tilde{y}\right)\right)} \\ &= \frac{1}{C(x)} \exp\left(-l\left(f(x), y\right)\right). \end{aligned}$$

Assuming a prior density $\mathsf{f}_{\mathsf{F}}(f) = \exp\left(-\lambda m \Omega[f]\right)$, by Bayes' theorem we have the posterior density

$$\begin{aligned} \mathsf{f}_{\mathsf{F}|Z^m=z}(f) &\propto \exp\left(-\sum_{i=1}^{m} l\left(f(x_i), y_i\right)\right) \exp\left(-\lambda m \Omega[f]\right) \\ &\propto \exp\left(-R_{\mathrm{emp}}[f, z] - \lambda \Omega[f]\right). \end{aligned}$$

The MAP estimate is that classifier f_{MAP} which maximizes the last expression, i.e., the mode of the posterior density. Taking the logarithm we see that the choice of a regularizer is comparable to the choice of the prior probability in the Bayesian framework and therefore reflects prior knowledge.

3 Kernels and Linear Classifiers

In practice we are often given a vectorial representation $x = \vec{x}$ of the objects. Using the identity feature mapping, i.e., $\mathbf{x} = \boldsymbol{\phi}(x) = \vec{x}$, results in classifiers linear in input space. Theoretically, however, any mapping into a high-dimensional feature space is conceivable. Hence, we call a classifier *nonlinear* in input space whenever a feature mapping different from the identity map is used.

Example 2.13 (Nonlinear classifiers) *Let $\mathcal{X} = \mathbb{R}^2$ and let the mapping $\boldsymbol{\phi} : \mathcal{X} \to \mathcal{K}$ be given by*

$$\boldsymbol{\phi}(\vec{x}) = \left((\vec{x})_1, (\vec{x})_2^2, (\vec{x})_1 (\vec{x})_2\right)'. \tag{2.15}$$

In Figure 2.3 (left) the mapping is applied to the unit square $[0, 1]^2$ and the resulting manifold in $\mathbb{R}^3$ is shown. Note that in this case the decision surface $\widetilde{X}_0(\mathbf{w})$

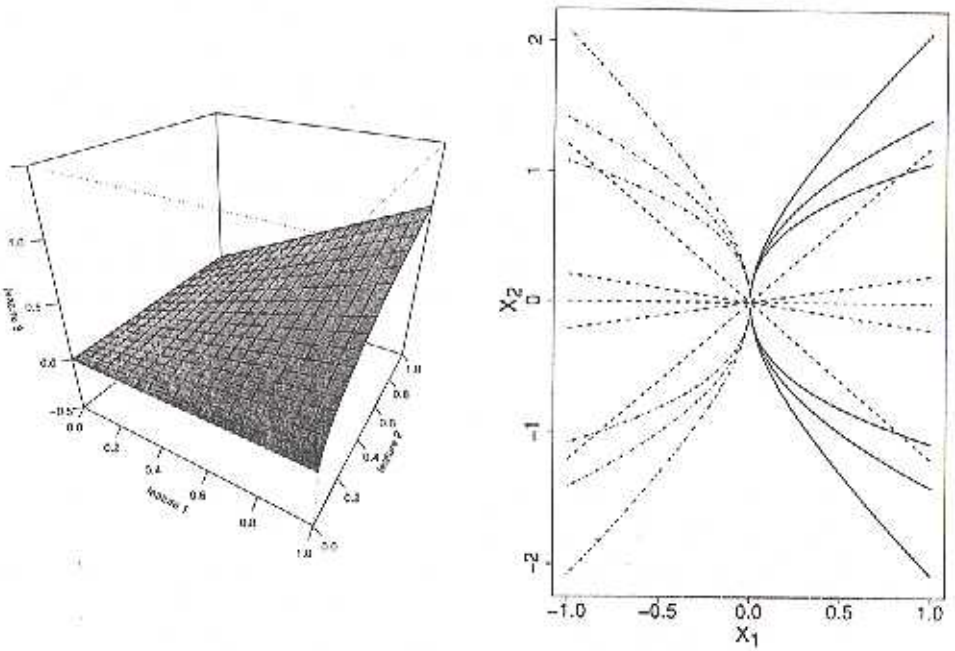

Figure 2.3 **(Left)** Mapping of the unit square $[0, 1]^2 \subset \mathbb{R}^2$ to the feature space $\mathcal{K} \subseteq \ell_2^3$ by equation (2.15). The mapped unit square forms a two-dimensional sub-manifold in $\mathbb{R}^3$ though $\dim(\mathcal{K}) = 3$. **(Right)** Nine different decision surfaces obtained by varying w_1 and w_3 in equation (2.16). The solid, dashed and dot-dashed lines result from varying w_3 for different values of $w_1 = -1, 0$ and $+1$, respectively.

in input space is given by

$$\widetilde{X}_0(\mathbf{w}) = \left\{ \vec{x} \in \mathbb{R}^2 \;\middle|\; w_1 (\vec{x})_1 + w_2 (\vec{x})_2^2 + w_3 (\vec{x})_1 (\vec{x})_2 = 0 \right\}, \tag{2.16}$$

whose solution is given by

$$(\vec{x})_2 = -\frac{w_3}{2w_2} \cdot (\vec{x})_1 \pm \frac{\sqrt{(\vec{x})_1 \left(w_3^2 (\vec{x})_1 - 4 w_1 w_2 \right)}}{2w_2}.$$

In Figure 2.3 (right) we have depicted the resulting decision surfaces for various choices of w_1 and w_3. Clearly, the decision surfaces are nonlinear functions although in feature space we are still dealing with linear functions.

As we assume $\boldsymbol{\phi}$ to be given we will call this the *explicit* way to non-linearize a linear classification model. We already mentioned in Section 2.2 that the number of dimensions, n, of the feature space has a great impact on the generalization ability of empirical risk minimization algorithms. Thus, one conceivable criterion for defining features ϕ_i is to seek a small set of basis functions ϕ_i which allow perfect discrimination between the classes in $\mathcal{X}$. This task is called *feature selection*.

Let us return to the primal perceptron learning algorithm mentioned in the last subsection. As we start at $\mathbf{w}_0 = \mathbf{0}$ and add training examples only when a mistake is committed by the current hypothesis, it follows that the each solution has to admit a representation of the form,

$$\mathbf{w}_t = \sum_{i=1}^{m} \alpha_i \boldsymbol{\phi}(x_i) = \sum_{i=1}^{m} \alpha_i \mathbf{x}_i \,. \tag{2.17}$$

Hence, instead of formulating the perceptron algorithm in terms of the n variables $(w_1, \ldots, w_n)' = \mathbf{w}$ we could learn the m variables $(\alpha_1, \ldots, \alpha_m)' = \boldsymbol{\alpha}$ which we call the *dual space* of variables. In the case of perceptron learning we start with $\boldsymbol{\alpha}_0 = \mathbf{0}$ and then employ the representation of equation (2.17) to update $\boldsymbol{\alpha}_t$ whenever a mistake occurs. To this end, we need to evaluate

$$y_j \langle \mathbf{x}_j, \mathbf{w}_t \rangle = y_j \left\langle \mathbf{x}_j, \sum_{i=1}^{m} \alpha_i \mathbf{x}_i \right\rangle = y_j \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}_j, \mathbf{x}_i \rangle$$

which requires only knowledge of the inner product function $\langle \cdot, \cdot \rangle$ between the mapped training objects $\mathbf{x}$. Further, for the classification of a novel test object x it suffices to know the solution vector $\boldsymbol{\alpha}_t$ as well as the inner product function, because

$$\langle \mathbf{x}, \mathbf{w}_t \rangle = \left\langle \mathbf{x}, \sum_{i=1}^{m} \alpha_i \mathbf{x}_i \right\rangle = \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle \,.$$

Definition 2.14 (Kernel) *Suppose we are given a feature mapping $\boldsymbol{\phi} : \mathcal{X} \to \mathcal{K} \subseteq \ell_2^n$. The* kernel *is the inner product function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ in $\mathcal{K}$, i.e., for all $x_i, x_j \in \mathcal{X}$,*

$$k(x_i, x_j) \stackrel{\text{def}}{=} \langle \boldsymbol{\phi}(x_i), \boldsymbol{\phi}(x_j) \rangle = \langle \mathbf{x}_i, \mathbf{x}_j \rangle \,.$$

Using the notion of a kernel k we can therefore formulate the *kernel perceptron* or *dual perceptron algorithm* as presented on page 324. Note that we can benefit

from the fact that, in each update step, we only increase the jth component of the expansion vector $\boldsymbol{\alpha}$ (assuming that the mistake occurred at the jth training point). This can change the real-valued output $\langle \mathbf{x}_i, \mathbf{w}_t \rangle$ at each mapped training object $\mathbf{x}_i$ by only one summand $y_j \langle \mathbf{x}_j, \mathbf{x}_i \rangle$ which requires just one evaluation of the kernel function with all training objects. Hence, by caching the real-valued outputs $\mathbf{o} \in \mathbb{R}^m$ at all training objects we see that the kernel perceptron algorithm requires exactly $2m$ memory units (for the storage of the vectors $\boldsymbol{\alpha}$ and $\mathbf{o}$) and is thus suited for large scale problems, i.e., $m \gg 1000$.

Definition 2.15 (Gram matrix) *Given a kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and a set $\boldsymbol{x} = (x_1, \ldots, x_m) \in \mathcal{X}^m$ of m objects in $\mathcal{X}$ we call the $m \times m$ matrix $\mathbf{G}$ with*

$$\mathbf{G}_{ij} \stackrel{\text{def}}{=} k\left(x_i, x_j\right) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{2.18}$$

the Gram matrix *of k at $\boldsymbol{x}$.*

By the above reasoning we see that the Gram matrix (2.18) and the m–dimensional vector of kernel evaluations between the training objects x_i and a new test object $x \in \mathcal{X}$ suffice for learning and classification, respectively. It is worth also mentioning that the Gram matrix and feature space are called the *kernel matrix* and *kernel space*, respectively, as well.

2.3.1 The Kernel Technique

The key idea of the kernel technique is to invert the chain of arguments, i.e., choose a kernel k rather than a mapping *before* applying a learning algorithm. Of course, not any symmetric function k can serve as a kernel. The necessary and sufficient conditions of $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ to be a kernel are given by Mercer's theorem. Before we rephrase the original theorem we give a more intuitive characterization of Mercer kernels.

Example 2.16 (Mercer's theorem) *Suppose our input space $\mathcal{X}$ has a finite number of elements, i.e., $\mathcal{X} = [x_1, \ldots, x_r]$. Then, the $r \times r$ kernel matrix $\mathbf{K}$ with $\mathbf{K}_{ij} = k\left(x_i, x_j\right)$ is by definition a symmetric matrix. Consider the eigenvalue decomposition of $\mathbf{K} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}'$, where $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_n) = \left(\mathbf{v}_1'; \ldots; \mathbf{v}_r'\right)$ is an $r \times n$ matrix such that $\mathbf{U}'\mathbf{U} = \mathbf{I}_n$, $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$, $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n > 0$ and $n \leq r$ being known as the rank of the matrix $\mathbf{K}$ (see also Theorem A.83 and*

Definition A.62). Now the mapping $\boldsymbol{\phi} : \mathcal{X} \to K \subseteq \ell_2^n$,

$$\boldsymbol{\phi}(x_i) = \boldsymbol{\Lambda}^{\frac{1}{2}} \mathbf{v}_i ,$$

leads to a Gram matrix $\mathbf{G}$ *given by*

$$\mathbf{G}_{ij} = \left\langle \boldsymbol{\phi}(x_i), \boldsymbol{\phi}(x_j) \right\rangle_K = \left(\boldsymbol{\Lambda}^{\frac{1}{2}} \mathbf{v}_i\right)' \left(\boldsymbol{\Lambda}^{\frac{1}{2}} \mathbf{v}_j\right) = \mathbf{v}_i' \boldsymbol{\Lambda} \mathbf{v}_j = \mathbf{K}_{ij} .$$

We have constructed a feature space K *and a mapping* $\boldsymbol{\phi}$ *into it purely from the kernel* k. *Note that* $\lambda_n > 0$ *is equivalent to assuming that* $\mathbf{K}$ *is positive semidefinite denoted by* $\mathbf{K} \geq 0$ *(see Definition A.40). In order to show that* $\mathbf{K} \geq 0$ *is also necessary for* k *to be a kernel, we assume that* $\lambda_n < 0$. *Then, the squared length of the point* $\sum_{i=1}^{r} \mathbf{U}_{in} \boldsymbol{\phi}(x_i) = \boldsymbol{\Lambda}^{\frac{1}{2}} \mathbf{U}' \mathbf{u}_n$ *is*

$$\left\| \boldsymbol{\Lambda}^{\frac{1}{2}} \mathbf{U}' \mathbf{u}_n \right\|^2 = \mathbf{u}_n' \mathbf{U} \boldsymbol{\Lambda} \mathbf{U}' \mathbf{u}_n = \mathbf{e}_n' \boldsymbol{\Lambda} \mathbf{e}_n = \lambda_n < 0 ,$$

which contradicts the geometry in an inner product space.

Mercer's theorem is an extension of this property, mainly achieved by studying the eigenvalue problem for integral equations of the form

$$\int_{\mathcal{X}} k(x, \tilde{x}) f(\tilde{x}) \, d\tilde{x} = \lambda f(x) ,$$

where k is a bounded, symmetric and positive semidefinite function.

Theorem 2.17 (Mercer's theorem) *Suppose* $k \in L_\infty(\mathcal{X} \times \mathcal{X})$ *is a symmetric function, i.e.,* $k(x, \tilde{x}) = k(\tilde{x}, x)$, *such that the integral operator* $T_k : L_2(\mathcal{X}) \to L_2(\mathcal{X})$ *given by*

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

is positive semidefinite, that is,

$$\int_{\mathcal{X}} \int_{\mathcal{X}} k(\tilde{x}, x) f(x) f(\tilde{x}) \, dx d\tilde{x} \geq 0 , \tag{2.19}$$

for all $f \in L_2(\mathcal{X})$. *Let* $\psi_i \in L_2(\mathcal{X})$ *be the eigenfunction of* T_k *associated with the eigenvalue* $\lambda_i \geq 0$ *and normalized such that* $\|\psi_i\|_2 = \int_{\mathcal{X}} \psi_i^2(x) \, dx = 1$, *i.e.,*

$$\forall x \in \mathcal{X} : \qquad \int_{\mathcal{X}} k(x, \tilde{x}) \psi_i(\tilde{x}) \, d\tilde{x} = \lambda_i \psi_i(x) .$$

Then

1. $(\lambda_i)_{i\in\mathbb{N}} \in \ell_1$,
2. $\psi_i \in L_\infty(\mathcal{X})$,
3. k can be expanded in a uniformly convergent series, i.e.,

$$k(x, \tilde{x}) = \sum_{i=1}^{\infty} \lambda_i \psi_i(x) \psi_i(\tilde{x}) \tag{2.20}$$

holds for all $x, \tilde{x} \in \mathcal{X}$.

The positivity condition (2.19) is equivalent to the positive semidefiniteness of $\mathbf{K}$ in Example 2.16. This has been made more precise in the following proposition.

Proposition 2.18 (Mercer Kernels) *The function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a* Mercer kernel *if, and only if, for each $r \in \mathbb{N}$ and $\mathbf{x} = (x_1, \ldots, x_r) \in \mathcal{X}^r$, the $r \times r$ matrix $\mathbf{K} = \left(k\left(x_i, x_j\right)\right)_{i,j=1}^{r}$ is positive semidefinite.*

Remarkably, Mercer's theorem not only gives necessary and sufficient conditions for k to be a kernel, but also suggests a constructive way of obtaining features ϕ_i from a given kernel k. To see this, consider the mapping $\boldsymbol{\phi}$ from $\mathcal{X}$ into ℓ_2

$$\boldsymbol{\phi}(x) = \left(\sqrt{\lambda_1}\psi_1(x), \sqrt{\lambda_2}\psi_2(x), \ldots\right)'. \tag{2.21}$$

By equation (2.20) we have for each $x, \tilde{x} \in \mathcal{X}$

$$k(x, \tilde{x}) = \sum_{i=1}^{\infty} \lambda_i \psi_i(x) \psi_i(\tilde{x}) = \sum_{i=1}^{\infty} \phi_i(x) \phi_i(\tilde{x}) = \langle \boldsymbol{\phi}(x), \boldsymbol{\phi}(\tilde{x}) \rangle .$$

The features ψ_i are called *Mercer features;* the mapping

$$\boldsymbol{\psi}(x) = (\psi_1(x), \psi_2(x), \ldots)'$$

is known as the *Mercer map;* the image $\mathcal{M}$ of $\boldsymbol{\psi}$ is termed *Mercer space.*

Remark 2.19 (Mahalanobis metric) *Consider kernels k such that $\dim(\mathcal{K}) = \dim(\mathcal{M}) < \infty$. In order to have equal inner products in feature space $\mathcal{K}$ and Mercer space $\mathcal{M}$, we need to redefine the inner product in $\mathcal{M}$, i.e.,*

$$\langle \mathbf{a}, \mathbf{b} \rangle_{\mathcal{M}} = \mathbf{a}' \boldsymbol{\Lambda} \mathbf{b},$$

where $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$. *This metric appears in the study of covariances of multidimensional Gaussians and is also known as the* Mahalanobis metric. *In fact, there is a very close connection between covariance functions for Gaussian processes and kernels which we will discuss in more depth in Chapter 3.*

2.3.2 Kernel Families

So far we have seen that there are two ways of making linear classifiers nonlinear in input space:

1. Choose a mapping ϕ which *explicitly* gives us a (Mercer) kernel k, or
2. Choose a Mercer kernel k which *implicitly* corresponds to a fixed mapping ϕ.

Though mathematically equivalent, kernels are often much easier to define and have the intuitive meaning of serving as a similarity measure between objects $x, \tilde{x} \in \mathcal{X}$. Moreover, there exist simple rules for designing kernels on the basis of given kernel functions.

Theorem 2.20 (Functions of kernels) *Let* $k_1 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *and* $k_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *be any two Mercer kernels. Then, the functions* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *given by*

1. $k(x, \tilde{x}) = k_1(x, \tilde{x}) + k_2(x, \tilde{x})$,
2. $k(x, \tilde{x}) = c \cdot k_1(x, \tilde{x})$, *for all* $c \in \mathbb{R}^+$,
3. $k(x, \tilde{x}) = k_1(x, \tilde{x}) + c$, *for all* $c \in \mathbb{R}^+$,
4. $k(x, \tilde{x}) = k_1(x, \tilde{x}) \cdot k_2(x, \tilde{x})$,
5. $k(x, \tilde{x}) = f(x) \cdot f(\tilde{x})$, *for any function* $f : \mathcal{X} \to \mathbb{R}$

are also Mercer kernels.

The proofs can be found in Appendix B.1. The real impact of these design rules becomes apparent when we consider the following corollary (for a proof see Appendix B.1).

Corollary 2.21 (Functions of kernels) *Let* $k_1 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *be any Mercer kernel. Then, the functions* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *given by*

1. $k(x, \tilde{x}) = (k_1(x, \tilde{x}) + \theta_1)^{\theta_2}$, *for all* $\theta_1 \in \mathbb{R}^+$ *and* $\theta_2 \in \mathbb{N}$,
2. $k(x, \tilde{x}) = \exp\left(\frac{k_1(x, \tilde{x})}{\sigma^2}\right)$, *for all* $\sigma \in \mathbb{R}^+$,

3. $k(x, \tilde{x}) = \exp\left(-\frac{k_1(x,x)-2k_1(x,\tilde{x})+k_1(\tilde{x},\tilde{x})}{2\sigma^2}\right)$, *for all* $\sigma \in \mathbb{R}^+$
4. $k(x, \tilde{x}) = \frac{k_1(x,\tilde{x})}{\sqrt{k_1(x,x)\cdot k_1(\tilde{x},\tilde{x})}}$

are also Mercer kernels.

It is worth mentioning that, by virtue of the fourth proposition of this corollary, it is possible to normalize data in feature space without performing the explicit mapping because, for the inner product after normalization, it holds that

$$k_{\text{norm}}(x, \tilde{x}) \stackrel{\text{def}}{=} \frac{k(x, \tilde{x})}{\sqrt{k(x,x)\cdot k(\tilde{x},\tilde{x})}} = \frac{1}{\sqrt{\|\mathbf{x}\|^2 \cdot \|\tilde{\mathbf{x}}\|^2}} \langle \mathbf{x}, \tilde{\mathbf{x}} \rangle = \left\langle \frac{\mathbf{x}}{\|\mathbf{x}\|}, \frac{\tilde{\mathbf{x}}}{\|\tilde{\mathbf{x}}\|} \right\rangle . \quad (2.22)$$

Kernels on Inner Product Spaces—Polynomial and RBF Kernels

If the input space $\mathcal{X}$ is already an N–dimensional inner product space ℓ_2^N we can use Corollary 2.21 to construct new kernels because, according to Example A.41 at page 219, the inner product function $\langle \cdot, \cdot \rangle_{\mathcal{X}}$ in $\mathcal{X}$ is already a Mercer kernel. In Table 2.1 some commonly used families of kernels on ℓ_2^N are presented. The last column gives the number of linearly independent features ϕ_i in the induced feature space $\mathcal{K}$.

The *radial basis function* (RBF) kernel has the appealing property that each linear combination of kernel functions of the training objects[9] $\boldsymbol{x} = (\vec{x}_1, \ldots, \vec{x}_m)$

$$f(\vec{x}) = \sum_{i=1}^{m} \alpha_i k(\vec{x}, \vec{x}_i) = \sum_{i=1}^{m} \alpha_i \exp\left(-\frac{\|\vec{x} - \vec{x}_i\|_{\mathcal{X}}^2}{2\sigma^2}\right), \quad (2.23)$$

can also be viewed as a density estimator in input space $\mathcal{X}$ because it effectively puts a Gaussian on each $\vec{x}_i$ and weights its contribution to the final density by α_i. Interestingly, by the third proposition of Corollary 2.21, the weighting coefficients α_i correspond directly to the expansion coefficients for a weight vector $\mathbf{w}$ in a classical linear model $f(\vec{x}) = \langle \phi(\vec{x}), \mathbf{w} \rangle$. The parameter σ controls the amount of smoothing, i.e., big values of σ lead to very flat and smooth functions f—hence it defines the unit on which distances $\|\vec{x} - \vec{x}_i\|$ are measured (see Figure 2.4). The *Mahalanobis* kernel differs from the standard RBF kernel insofar as each axis of the input space $\mathcal{X} \subseteq \ell_2^N$ has a separate smoothing parameter, i.e., a

9 In this subsection we use $\vec{x}$ to denote the N–dimensional vectors in input space. Note that $\mathbf{x} := \phi(\vec{x})$ denotes a mapped input object (vector) $\vec{x}$ in feature space $\mathcal{K}$.

Name	Kernel function	$\dim(\mathcal{K})$
pth degree polynomial	$k(\vec{u},\vec{v}) = (\langle \vec{u},\vec{v}\rangle_{\mathcal{X}})^p$ $p \in \mathbb{N}^+$	$\binom{N+p-1}{p}$
complete polynomial	$k(\vec{u},\vec{v}) = (\langle \vec{u},\vec{v}\rangle_{\mathcal{X}} + c)^p$ $c \in \mathbb{R}^+,\ p \in \mathbb{N}^+$	$\binom{N+p}{p}$
RBF kernel	$k(\vec{u},\vec{v}) = \exp\left(-\frac{\|\vec{u}-\vec{v}\|_{\mathcal{X}}^2}{2\sigma^2}\right)$ $\sigma \in \mathbb{R}^+$	∞
Mahalanobis kernel	$k(\vec{u},\vec{v}) = \exp\left(-(\vec{u}-\vec{v})'\,\boldsymbol{\Sigma}\,(\vec{u}-\vec{v})\right)$ $\boldsymbol{\Sigma} = \operatorname{diag}\left(\sigma_1^{-2},\ldots,\sigma_N^{-2}\right)$, $\sigma_1,\ldots,\sigma_N \in \mathbb{R}^+$	∞

Table 2.1 List of kernel functions over ℓ_2^N. The dimensionality of the input space is N.

separate scale onto which differences on this axis are viewed. By setting $\sigma_i \to \infty$ we are able to eliminate the influence of the ith feature in input space. We shall see in Section 3.2 that inference over these parameters is made in the context of *automatic relevance determination* (ARD) of the features in input space (see also Example 3.12). It is worth mentioning that RBF kernels map the input space onto the surface of an infinite dimensional hypersphere because by construction $\|\phi(\vec{x})\| = \sqrt{k(\vec{x},\vec{x})} = 1$ for all $\vec{x} \in \mathcal{X}$. Finally, by using RBF kernels we have automatically chosen a classification model which is shift invariant, i.e., translating the whole input space $\mathcal{X}$ by some fixed vector $\vec{a}$ does not change anything because

$$\forall \vec{a} \in \mathcal{X}: \quad \|(\vec{x}+\vec{a}) - (\vec{x}_i+\vec{a})\|^2 = \|\vec{x}+\vec{a} \quad \vec{x}_i-\vec{a}\|^2 = \|\vec{x}-\vec{x}_i\|^2 .$$

The most remarkable advantage in using these kernels is the saving in computational effort, e.g., to calculate the inner product for pth degree complete polynomial kernels we need $\mathcal{O}(N+p)$ operations whereas an explicit mapping would require calculations of order $\mathcal{O}(\exp(p\ln(N/p)))$. Further, for radial basis function kernels, it is very difficult to perform the explicit mapping.

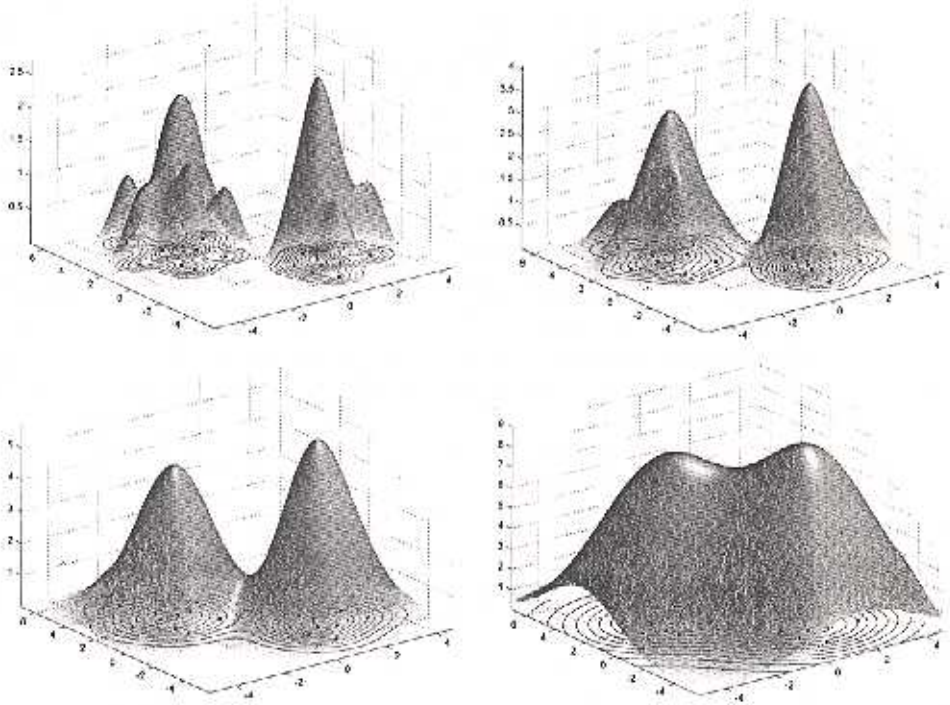

Figure 2.4 The real-valued function $f(\vec{x})$ for $m = 20$ training points $\vec{x} \in \mathbb{R}^2$ with $\alpha = 1$ (see equation (2.23)) for varying values of σ (from upper left to lower right $\sigma = 0.5$, $\sigma = 0.7$, $\sigma = 1.0$ and $\sigma = 2.0$). From the contour plot it can be seen that by increasing σ the contribution of single points to the final density vanishes. Further, for bigger values of σ the resulting surface is smoother. For visualization purposes the surface $\{\vec{x} \mid f(\vec{x}) = 0\}$ is made transparent.

Example 2.22 (Polynomial kernel) *Consider the pth degree polynomial kernel as given in Table 2.1. In order to obtain explicit features $\phi : \mathcal{X} \to \mathbb{R}$ let us expand the kernel function as follows*[10]

$$(\langle \vec{u}, \vec{v} \rangle_{\mathcal{X}})^p = \left(\sum_{i=1}^{N} u_i v_i \right)^p = \left(\sum_{i_1=1}^{N} u_{i_1} v_{i_1} \right) \cdots \left(\sum_{i_p=1}^{N} u_{i_p} v_{i_p} \right)$$

10 For notational brevity, in this example we denote the i--th component of the vector $\vec{u} \in \mathcal{X}$ and $\vec{v} \in \mathcal{X}$ by u_i and v_i, respectively.

$$= \sum_{i_1=1}^{N} \cdots \sum_{i_p=1}^{N} \underbrace{\left(u_{i_1} \cdots u_{i_p}\right)}_{\phi_{\mathbf{i}}(\vec{u})} \cdot \underbrace{\left(v_{i_1} \cdots v_{i_p}\right)}_{\phi_{\mathbf{i}}(\vec{v})} = \langle \boldsymbol{\phi}(\vec{u}), \boldsymbol{\phi}(\vec{v}) \rangle \ .$$

Although it seems that there are N^p different features we see that two index vectors $\mathbf{i}_1$ and $\mathbf{i}_2$ lead to the same feature $\phi_{\mathbf{i}_1} = \phi_{\mathbf{i}_2}$ if they contain the same distinct indices the same number of times but at different positions, e.g., $\mathbf{i}_1 = (1,1,3)$ and $\mathbf{i}_2 = (1,3,1)$ both lead to $\phi(\vec{u}) = u_1 u_1 u_3 = u_1^2 u_3$. One method of computing the number of different *features ϕ is to index them by an N–dimensional exponent vector $\mathbf{r} = (r_1, \ldots, r_N) \in \{0, \ldots, p\}^N$, i.e., $\phi_{\mathbf{r}}(\vec{u}) = u_1^{r_1} \cdot \cdots \cdot u_N^{r_N}$. Since there are exactly p summands we know that each admissible exponent vector $\mathbf{r}$ must obey $r_1 + \cdots + r_N = p$. The* number *of different exponent vectors $\mathbf{r}$ is thus exactly given by*[11]

$$\binom{N+p-1}{p},$$

and for each admissible exponent vector $\mathbf{r}$ there are exactly[12]

$$\frac{p!}{r_1! \cdot \cdots \cdot r_N!}$$

different index vectors $\mathbf{i} \in \{1, \ldots, N\}^p$ leading to $\mathbf{r}$. Hence the $\mathbf{r}$th feature is given by

$$\phi_{\mathbf{r}}(\vec{u}) = \sqrt{\frac{p!}{r_1! \cdot \cdots \cdot r_N!}} \cdot u_1^{r_1} \cdot \cdots \cdot u_N^{r_N} \ .$$

Finally note that the complete polynomial kernel in Table 2.1 is a pth degree polynomial kernel in an $N+1$–dimensional input space by the following identity

$$(\langle \vec{u}, \vec{v} \rangle + c)^p = \left(\left(\vec{u}, \sqrt{c} \right), \left(\vec{v}, \sqrt{c} \right) \right)^p \ ,$$

11 This problem is known as the *occupancy problem*: Given p balls and N cells, how many different configurations of occupancy numbers $r_1, \ldots, r_N$ whose sum is exactly p exist? (see Feller (1950) for results).

12 To see this note that we have first to select r_1 indices $j_1, \ldots, j_{r_1}$ and set $i_{j_1} = \cdots = i_{j_{r_1}} = 1$. From the remaining $p - r_1$ indices select r_2 indices and set them all to 2, etc. Thus, the total number of different index vectors $\mathbf{i}$ leading to the same exponent vector $\mathbf{r}$ equals

$$\binom{p}{r_1}\binom{p-r_1}{r_2} \cdot \cdots \cdot \binom{p-r_1-\cdots-r_{N-2}}{r_{N-1}} = \frac{p!}{r_1! \cdot \cdots \cdot r_N!},$$

which is valid because $r_1 + \cdots + r_N = p$ (taken from Feller (1950)).

where we use the fact that $c \geq 0$. This justifies the number of dimensions of feature space given in the third column of Table 2.1.

Kernels on Strings

One of the greatest advantages of kernels is that they are not limited to vectorial objects $\vec{x} \in \mathcal{X}$ but that they are applicable to virtually any kind of object representation. In this subsection we will demonstrate that it is possible to efficiently formulate computable kernels on strings. An application of string kernels is in the analysis of DNA sequences which are given as strings composed of the symbols[13] A, T, G, C. Another interesting use of kernels on strings is in the field of text categorization and classification. Here we treat each document as a sequence or string of letters. Let us start by formalizing the notion of a string.

Definition 2.23 (Strings and alphabets) *An* alphabet Σ *is a finite collection of symbols called characters. A* string *is a finite sequence $\boldsymbol{u} = (u_1, \ldots, u_r)$ of characters from an alphabet Σ. The symbol Σ^* denotes the set of all strings of any length, i.e., $\Sigma^* \stackrel{\text{def}}{=} \cup_{i=0}^{\infty} \Sigma^i$. The number $|\boldsymbol{u}|$ of symbols in a string $\boldsymbol{u} \in \Sigma^*$ is called the* length *of the string. Given two strings $\boldsymbol{u} \in \Sigma^*$ and $\boldsymbol{v} \in \Sigma^*$, the symbol $\boldsymbol{uv} \stackrel{\text{def}}{=} (u_1, \ldots, u_{|\boldsymbol{u}|}, v_1, \ldots, v_{|\boldsymbol{v}|})$ denotes the* concatenation *of the two strings.*

Definition 2.24 (Subsequences and substrings) *Given a string $\boldsymbol{u} \in \Sigma^*$ and an index vector $\mathbf{i} = (i_1, \ldots, i_r)$ such that $1 \leq i_1 < \cdots < i_r \leq |\boldsymbol{u}|$, we denote by $\boldsymbol{u}[\mathbf{i}]$ the* subsequence $(u_{i_1}, \ldots, u_{i_r})$. *The index vector $(1, \ldots, r)$ is abbreviated by $1:r$. Given two strings $\boldsymbol{v} \in \Sigma^*$ and $\boldsymbol{u} \in \Sigma^*$ where $|\boldsymbol{u}| \geq |\boldsymbol{v}|$ we define the index set $I_{\boldsymbol{v},\boldsymbol{u}} \stackrel{\text{def}}{=} \{i : (i + |\boldsymbol{v}| - 1) \mid i \in \{1, \ldots, |\boldsymbol{u}| - |\boldsymbol{v}| + 1\}\}$, i.e., the set of all consecutive sequences of length $|\boldsymbol{v}|$ in $|\boldsymbol{u}|$. Then the string $\boldsymbol{v}$ is said to be a* substring *of $\boldsymbol{u}$ if there exists an index vector $\mathbf{i} \in I_{\boldsymbol{v},\boldsymbol{u}}$ such that $\boldsymbol{v} = \boldsymbol{u}[\mathbf{i}]$. The* length $l(\mathbf{i})$ *of an index vector is defined by $i_{|\boldsymbol{v}|} - i_1 + 1$, i.e., the total extent of the subsequence (substring) $\boldsymbol{v}$ in the string $\boldsymbol{u}$.*

In order to derive kernels on strings, it is advantageous to start with the explicit mapping $\phi : \Sigma^* \to \mathcal{K}$ and then make sure that the resulting inner product function $\langle \phi(\cdot), \phi(\cdot) \rangle$ is easy to compute. By the finiteness of the alphabet Σ, the set Σ^* is countable and we can therefore use it to index the features ϕ.

13 These letters correspond to the four bases *Adenine, Thymine, Guanine* and *Cytosine*.

The most trivial feature set and corresponding kernel are obtained if we consider binary features ϕ_u that indicate whether the given string matches u or not,

$$\phi_u(v) = 1_{u=v} \quad \Leftrightarrow \quad k(u,v) = \begin{cases} 1 & \text{if } u = v \\ 0 & \text{otherwise} \end{cases}.$$

Though easy to compute, this kernel is unable to measure the similarity to any object (string) not in the training sample and hence would not be useful for learning.

A more commonly used feature set is obtained if we assume that we are given a *lexicon* $B = \{b_1, \ldots, b_n\} \subset \Sigma^*$ of possible substrings which we will call *words*. We compute the number of times the ith substring b_i appears within a given string (document). Hence, the so-called *bag-of-words kernel* is given by

$$\phi_b(v) = \beta_b \cdot \sum_{i \in I_{b,v}} 1_{b=v[i]} \quad \Leftrightarrow \quad k_B(u,v) = \sum_{b \in B} \beta_b^2 \sum_{i \in I_{b,u}} \sum_{j \in I_{b,v}} 1_{b=u[i]=v[j]}, \tag{2.24}$$

which can be efficiently computed if we assume that the data is preprocessed such that only the indices of the words occurring in a given string are stored. The coefficients β_b allow the weighting of the importance of words $b \in B$ to differ. A commonly used heuristic for the determination of the β_b is the use of the *inverse-document-frequency* (IDF) which is given by the logarithm of the inverse probability that the substring (word) b appears in a randomly chosen string (document).

The kernel given in equation (2.24) has the disadvantage of requiring a fixed lexicon $B \subset \Sigma^*$ which is often difficult to define *a-priori*. This is particularly true when dealing with strings *not* originating from natural languages. If we fix the maximum length, r, of substrings considered and weight the feature ϕ_b by $\lambda^{|b|}$, i.e., for $\lambda \in (0,1)$ we emphasize short substrings whereas for $\lambda > 1$ the weight of longer substrings increases, we obtain

$$\phi_b(v) = \lambda^{|b|} \sum_{i \in I_{b,v}} 1_{b=v[i]} \quad \Leftrightarrow \quad k_r(u,v) = \sum_{s=1}^{r} \lambda^{2s} \sum_{b \in \Sigma^s} \sum_{i \in I_{b,u}} \sum_{j \in I_{b,v}} 1_{b=u[i]=v[j]}, \tag{2.25}$$

which can be computed using the following recursion (see Appendix B.2)

$$k_r(u_1 u, v) = \begin{cases} 0 & \text{if } |u_1 u| = 0 \\ k_r(u,v) + \sum_{j=1}^{|v|} \lambda^2 \cdot k'_r(u_1 u, v) & \text{otherwise} \end{cases}, \tag{2.26}$$

$$k'_r(u_1 u, v_1 v) = \begin{cases} 0 & \text{if } r = 0 \\ 0 & \text{if } |u_1 u| = 0 \text{ or } |v_1 v| = 0 \\ 0 & \text{if } u_1 \neq v_1 \\ \left(1 + \lambda^2 \cdot k'_{r-1}(u, v)\right) & \text{otherwise} \end{cases} \quad (2.27)$$

Since the recursion over k_r invokes at most $|v|$ times the recursion over k'_r (which terminates after at most r steps) and is invoked itself exactly $|u|$ times, the computational complexity of this string kernel is $\mathcal{O}(r \cdot |u| \cdot |v|)$.

One of the disadvantages of the kernels given in equations (2.24) and (2.25) is that each feature requires a perfect match of the substring b in the given string $v \in \Sigma^*$. In general, strings can suffer from deletion and insertion of symbols, e.g., for DNA sequences it can happen that a few bases are inserted somewhere in a given substring b. Hence, rather than requiring b to be a substring we assume that $\phi_b(v)$ only measures how often b is a subsequence of v and penalizes the non-contiguity of b in v by using the length $l(\mathbf{i})$ of the corresponding index vector $\mathbf{i}$, i.e.,

$$\phi_b(v) = \sum_{\{\mathbf{i} | b = v[\mathbf{i}]\}} \lambda^{l(\mathbf{i})} \quad \Leftrightarrow \quad k_r(u, v) = \sum_{b \in \Sigma^r} \sum_{\{\mathbf{i} | b = u[\mathbf{i}]\}} \sum_{\{\mathbf{j} | b = v[\mathbf{j}]\}} \lambda^{l(\mathbf{i}) + l(\mathbf{j})} \quad (2.28)$$

This kernel can efficiently be computed by applying the the following recursion formula (see Appendix B.2)

$$k_r(u u_s, v) = \begin{cases} 0 & \text{if } \min(|u u_s|, |v|) < r \\ k_r(u, v) + \lambda^2 \sum_{\{t | v_t = u_s\}} k'_{r-1}(u, v[1 : (t-1)]) \end{cases} \quad (2.29)$$

$$k'_r(u u_s, v) = \begin{cases} 0 & \text{if } \min(|u u_s|, |v|) < r \\ 1 & \text{if } r = 0 \\ \lambda \cdot k'_r(u, v) + \lambda^2 \sum_{\{t | v_t = u_s\}} \lambda^{|v| - t} k'_{r-1}(u, v[1 : (t-1)]) \end{cases} \quad (2.30)$$

Clearly, the recursion for k_r is invoked exactly $|u|$ times by itself and each time invokes at most $|v|$ times the recursive evaluation of k'_r. The recursion over k'_r is invoked at most r times itself and invokes at most $|v|$ times the recursion over k'_{r-1}. As a consequence the computational complexity of this algorithm is $\mathcal{O}\left(r \cdot |u| \cdot |v|^2\right)$. It can be shown, however, that with simple caching it is possible to reduce the complexity further to $\mathcal{O}(r \cdot |u| \cdot |v|)$.

Remark 2.25 (Ridge Problem) *The kernels (2.25) and (2.28) lead to the so-called* ridge problem *when applied to natural language documents, i.e., different documents $u \in \Sigma^*$ and $v \in \Sigma^*$ map to almost orthogonal features $\phi(u)$ and $\phi(v)$. Thus, the Gram matrix has a dominant diagonal (see Figure 2.5) which is prob-*

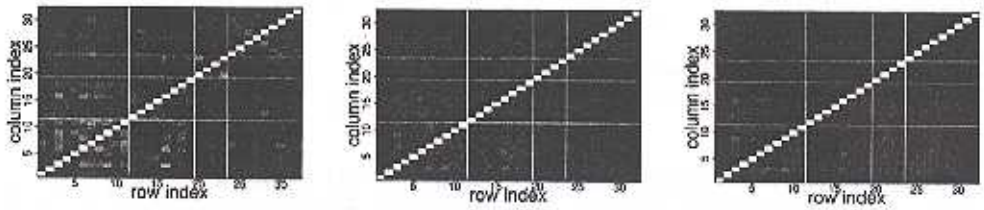

Figure 2.5 Intensity plots of the normalized Gram matrices when applying the string kernels (2.24), (2.25) and (2.28) (from left to right) to 32 sentences taken from this chapter with $n = 5$ and $\lambda = 0.5$. 11, 8, 4 and 9 sentences were taken from Section 2.2, Subsection 2.2.2, Section 2.3 and Subsection 2.3.1, respectively. For the sake of clarity, white lines are inserted to indicate the change from one section to another section.

lematic because each new test document x is likely to have a kernel value $k(x, x_i)$ close to zero. In order to explain this we notice that a document $\boldsymbol{u} \in \Sigma^*$ has at least $|\boldsymbol{u}| - r + 1$ matches of contiguous substrings with itself, i.e., all substrings $\boldsymbol{u}[i : (i + r - 1)]$ for all $i \in \{1, \ldots, |\boldsymbol{u}| - r + 1\}$. However, even if two documents $\boldsymbol{u} \in \Sigma^*$ and $\boldsymbol{v} \in \Sigma^*$ share all words $\boldsymbol{b} \in \Sigma^r$ of length r (on average) but in different orders, we have approximately $\frac{|\boldsymbol{u}|}{r}$ matches (assuming $|\boldsymbol{u}| \approx |\boldsymbol{v}|$). Therefore the difference $\left((|\boldsymbol{u}| - r) - \frac{|\boldsymbol{u}|}{r}\right) \cdot \lambda^r$ between diagonal and off-diagonal elements of the Gram matrix becomes systematically larger with increasing subsequence length r.

Kernels from Probabilistic Models of the Data

A major disadvantage of the two kernel families presented so far is that they are limited to a fixed representation of objects, x, i.e., vectorial data or strings. In order to overcome this limitation, Jaakkola and Haussler introduced the so-called Fisher kernel. The idea of the Fisher kernel is to use a probabilistic model of the input data, x, to derive a similarity measure between two data items. In order to achieve this, let us assume that the object generating probability measure P_X can be written as a mixture, i.e., there exists a vector $\boldsymbol{\theta} = (\theta_1; \ldots; \theta_r; \boldsymbol{\pi})$ such that[14]

$$\mathsf{P}_\mathsf{X}(x) = \mathsf{P}_\mathsf{X}^{\boldsymbol{\theta}}(x) = \sum_{i=1}^{r} \mathsf{P}_{\mathsf{X}|\mathsf{M}=i}^{\theta_i}(x) \cdot \underbrace{\mathsf{P}_\mathsf{M}(i)}_{\pi_i} = \sum_{i=1}^{r} \pi_i \cdot \mathsf{P}_{\mathsf{X}|\mathsf{M}=i}^{\theta_i}(x) \;, \tag{2.31}$$

14 With a slight abuse of notation, we always use P_X even if X is a continuous random variable possessing a density f_X. In this case we have to replace P_X by f_X and $\mathsf{P}_{\mathsf{X}|\mathsf{M}=i}$ by $\mathsf{f}_{\mathsf{X}|\mathsf{M}=i}$ but the argument would not change.

where the measure $\mathbf{P}^{\theta_i}_{X|M=i}$ is parameterized by θ_i only. In the search for the most plausible mixture components θ_{ML} (given a set $x \in \mathcal{X}^m$ of m training objects) the Fisher score and the Fisher information matrix play a major role.

Definition 2.26 (Fisher score and Fisher information matrix) *Given a parameterized family $\mathcal{P}_Q$ of probability measures $\mathbf{P}^{\theta}_X$ over the space $\mathcal{X}$ and a parameter vector $\tilde{\theta} \in Q$ the function*

$$\mathbf{f}_{\tilde{\theta}}(x) \stackrel{\text{def}}{=} \left.\frac{\partial \ln\left(\mathbf{P}^{\theta}_X(x)\right)}{\partial \theta}\right|_{\theta=\tilde{\theta}}$$

is called the Fisher score *of x at $\tilde{\theta}$. Further, the matrix*

$$I_{\tilde{\theta}} \stackrel{\text{def}}{=} \mathbf{E}_X\left[\mathbf{f}_{\tilde{\theta}}(X)\left(\mathbf{f}_{\tilde{\theta}}(X)\right)'\right] \tag{2.32}$$

is called Fisher information matrix *at $\tilde{\theta}$. Note that the expectation in equation (2.32) is w.r.t. $\mathbf{P}^{\tilde{\theta}}_X$.*

Now, given an estimate $\hat{\theta} \in Q$ of the parameter vector θ—probably obtained by using unlabeled data $\{x_1, \ldots, x_M\}$, where $M \gg m$—let us consider the *Fisher score mapping* in the $|\theta|$–dimensional feature space $\mathcal{K}$, i.e.,

$$\phi_{\hat{\theta}}(x) = \mathbf{f}_{\hat{\theta}}(x) \,. \tag{2.33}$$

Interestingly, we see that the features ϕ associated with π_i measure the amount by which the ith mixture component $\mathbf{P}_{X|M=i}$ contributes to the generation of the pattern x, i.e.,

$$\frac{\partial \ln\left(\mathbf{P}^{\theta}_X(x)\right)}{\partial \pi_j} = \frac{\partial \ln\left(\sum_{i=1}^{r} \pi_i \mathbf{P}^{\theta_i}_{X|M=i}(x)\right)}{\partial \pi_j} = \frac{\mathbf{P}^{\theta_j}_{X|M=j}(x)}{\sum_{i=1}^{r} \pi_i \mathbf{P}^{\theta_i}_{X|M=i}(x)} = \frac{\mathbf{P}^{\theta_j}_{X|M=j}(x)}{\mathbf{P}^{\theta}_X(x)} \,.$$

As a consequence, these features allow a good separation of all regions of the input space $\mathcal{X}$ in which the mixture measure (2.31) is high for exactly one component only. Hence, using the Fisher score $\mathbf{f}_{\theta}(x)$ as a vectorial representation of x provides a principled way of obtaining kernels from a generative probabilistic model of the data.

Definition 2.27 (Fisher kernel) *Given a parameterized family $\mathcal{P}$ of probability measures $\mathbf{P}_X^\theta$ over the input space $\mathcal{X}$ and a parameter vector $\theta \in Q$ the function*

$$k(x, \tilde{x}) = (\mathbf{f}_\theta(x))' \, I_\theta^{-1} \mathbf{f}_\theta(\tilde{x})$$

is called the Fisher kernel. *The* naive Fisher kernel *is the simplified function*

$$k(x, \tilde{x}) = (\mathbf{f}_\theta(x))' \, \mathbf{f}_\theta(\tilde{x}) \, .$$

This assumes that the Fisher information matrix I_θ is the identity matrix I.

The naive Fisher kernel is practically more relevant because the computation of the Fisher information matrix is very time consuming and sometimes not even analytically possible. Note, however, that not only do we need a probability model $\mathbf{P}_X^\theta$ of the data but also the model $\mathcal{P} \supset \mathbf{P}_X^\theta$ of probability measures.

Example 2.28 (Fisher kernel) *Let us assume that the measures $\mathbf{P}_{X|M=i}$ belong to the exponential family, i.e., their density can be written as*

$$\mathbf{f}_{X|M=i}^{\theta_i}(x) = a_i(\theta_i) \cdot c_i(x) \cdot \exp\left(\theta_i' \tau_i(x)\right) \, ,$$

where $c_i : \mathcal{X} \to \mathbb{R}$ is a fixed function, $\tau_i : \mathcal{X} \to \mathbb{R}^{n_i}$ is known as a sufficient statistic *of x and $a_i : \mathbb{R}^{n_i} \to \mathbb{R}$ is a normalization constant. Then the value of the features ϕ_{θ_j} associated with the jth parameter vector θ_j are given by*

$$\begin{aligned} \frac{\partial \ln\left(\mathbf{f}_X^\theta(x)\right)}{\partial \theta_j} &= \frac{1}{\mathbf{f}_X^\theta(x)} \cdot \frac{\partial\left(\sum_{i=1}^{r} \mathbf{P}_M(i) \cdot a_i(\theta_i) \cdot c_i(x) \cdot \exp\left(\theta_i' \tau_i(x)\right)\right)}{\partial \theta_j} \\ &= \frac{\mathbf{f}_{X|M=j}^{\theta_j}(x) \, \mathbf{P}_M(j)}{\mathbf{f}_X^\theta(x)} \left(\underbrace{\frac{\frac{\partial a_j(\theta_j)}{\partial \theta_j}}{a_j(\theta_j)}}_{\text{independent of } x} + \tau_j(x) \right) . \end{aligned}$$

Let us consider the contribution of the features ϕ_{θ_j} at objects $x, \tilde{x} \in \mathcal{X}$ for which[15]

$$\frac{\mathbf{f}_{X|M=j}^{\theta_j}(x)}{\mathbf{f}_X^\theta(x)} \approx \frac{\mathbf{f}_{X|M=j}^{\theta_j}(\tilde{x})}{\mathbf{f}_X^\theta(\tilde{x})}$$

15 If this relation does not hold then the features associated with x_j already allow good discrimination.

and, additionally, assume that $\mathbf{P}_M$ *is the uniform measure. We see that*

$$\left\langle \phi_{\theta_j}(x), \phi_{\theta_j}(\tilde{x}) \right\rangle \propto \left(\tau_j(x)\right)' \tau_j(\tilde{x}) ,$$

that is, we effectively consider the sufficient statistic $\tau_j(x)$ *of the* j*th mixture component measure as a vectorial representation of our data.*

2.3.3 The Representer Theorem

We have seen that kernels are a powerful tool that enrich the applicability of linear classifiers by a large extent. Nonetheless, apart from the solution of the perceptron learning algorithm it is not yet clear when this method can successfully be applied, i.e., for which learning algorithms $\mathcal{A} : \cup_{m=1}^{\infty} \mathcal{Z}^m \to \mathcal{F}$ the solution $\mathcal{A}(z)$ admits a representation of the form

$$(\mathcal{A}(z))(\cdot) = \sum_{i=1}^{m} \alpha_i k(x_i, \cdot) . \tag{2.34}$$

Before identifying this class of learning algorithms we introduce a purely functional analytic point of view on kernels. We will show that each Mercer kernel automatically defines a *reproducing kernel Hilbert space* (RKHS) of functions as given by equation (2.34). Finally, we identify the class of cost functions whose solution has the form (2.34).

Reproducing Kernel Hilbert Spaces

Suppose we are given a Mercer kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Then let $\mathcal{F}_0$ be the linear space of real-valued functions on $\mathcal{X}$ generated by the functions $\{k(x, \cdot) \mid x \in \mathcal{X}\}$. Consider any two functions $f(\cdot) = \sum_{i=1}^{r} \alpha_i k(x_i, \cdot)$ and $g(\cdot) = \sum_{j=1}^{s} \beta_j k(\tilde{x}_j, \cdot)$ in $\mathcal{F}_0$ where $\boldsymbol{\alpha} \in \mathbb{R}^r$, $\boldsymbol{\beta} \in \mathbb{R}^s$ and $x_i, \tilde{x}_j \in \mathcal{X}$. Define the inner product $\langle f, g \rangle$ between f and g in $\mathcal{F}_0$ as

$$\langle f, g \rangle \stackrel{\text{def}}{=} \sum_{i=1}^{r} \sum_{j=1}^{s} \alpha_i \beta_j k(x_i, \tilde{x}_j) = \sum_{j=1}^{s} \beta_j f(\tilde{x}_j) = \sum_{i=1}^{r} \alpha_i g(x_i) , \tag{2.35}$$

where the last equality follows from the symmetry of the kernel k. Note that this inner product $\langle \cdot, \cdot \rangle$ is independent of the representation of the function f and g because changing the representation of f, i.e., changing r, $\boldsymbol{\alpha}$ and $\{x_1, \ldots, x_r\}$, would not change $\sum_{j=1}^{s} \beta_j f(\tilde{x}_j)$ (similarly for g). Moreover, we see that

1. $\langle f, g\rangle = (g, f)$ for all functions $f, g \in \mathcal{F}_0$,

2. $\langle cf + dg, h\rangle = c\,(f, h\rangle + d\,(g, h\rangle$ for all functions $f, g, h \in \mathcal{F}_0$ and all $c, d \in \mathbb{R}$,

3. $\langle f, f\rangle = \sum_{i=1}^{r} \sum_{j=1}^{r} \alpha_i \alpha_j k\left(x_i, x_j\right) \geq 0$ for all functions $f \in \mathcal{F}_0$ because k is a Mercer kernel.

It still remains to established that $\langle f, f\rangle = 0$ implies that $f = 0$. To show this we need first the following important *reproducing property*: For all functions $f \in \mathcal{F}_0$ and all $x \in \mathcal{X}$

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

which follows directly from choosing $s = 1$, $\beta_1 = 1$ and $\tilde{x}_1 = x$ in (2.35)—hence $g\,(\cdot) = k\,(x, \cdot)$. Now using the Cauchy-Schwarz inequality (see Theorem A.106 and preceding comments) we know that

$$0 \leq (f\,(x))^2 = (\langle f, k\,(x, \cdot)\rangle)^2 \leq (f, f)\,\underbrace{\langle k\,(x, \cdot)\,, k\,(x, \cdot)\rangle}_{k(x,x)}\,, \tag{2.37}$$

which shows that $\langle f, f\rangle = 0$ only if $f\,(x) = 0$ for all $x \in \mathcal{X}$, i.e., $f = 0$.

Finally, let us consider any Cauchy sequence $(f_r)_{r\in\mathbb{N}}$ of functions in $\mathcal{F}_0$. Then, by virtue of equation (2.37), we know that, for all $r, s \in \mathbb{N}$, $(f_r\,(x) - f_s\,(x))^2 \leq \|f_r - f_s\|^2 k\,(x, x)$ and hence $(f_r)_{r\in\mathbb{N}}$ converges toward some real-valued function f on $\mathcal{X}$. It is possible to complete $\mathcal{F}_0$ by adding the limits of all Cauchy sequences to it, extending it and its inner product to a slightly larger class $\mathcal{F} \subseteq \mathbb{R}^{\mathcal{X}}$. Thus, we have shown that each kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ defines a Hilbert space $\mathcal{F}$ of real-valued functions over $\mathcal{X}$ which has the reproducing property (2.36), i.e., the value of the function f at x is "reproduced" by the inner product of f with $k\,(x, \cdot)$. The full power of this consideration is expressed in the following theorem.

Theorem 2.29 (Representer theorem) *Let k be a Mercer kernel on $\mathcal{X}$, $z \in (\mathcal{X} \times \mathcal{Y})^m$ be a training sample and $g_{\text{emp}} : (\mathcal{X} \times \mathcal{Y} \times \mathbb{R})^m \to \mathbb{R} \cup \{\infty\}$ be any arbitrary but fixed function. Let $g_{\text{reg}} : \mathbb{R} \to [0, \infty)$ be any strictly monotonically increasing function. Define $\mathcal{F}$ as the RKHS induced by k. Then any $f \in \mathcal{F}$ minimizing the regularized risk*

$$R_{\text{reg}}\,[f, z] = g_{\text{emp}}\left(((x_i, y_i, f\,(x_i)))_{i\in\{1,\ldots,m\}}\right) + g_{\text{reg}}\,(\|f\|)\,, \tag{2.38}$$

admits a representation of the form

$$f(\cdot) = \sum_{i=1}^{m} \alpha_i k(x_i, \cdot) \qquad \alpha \in \mathbb{R}^m . \tag{2.39}$$

The proof is given in Appendix B.3. It elucidates once more the advantage of kernels: Apart from limiting the computational effort in application, they allow for a quite general class of learning algorithms (characterized by the minimization of a functional of the form (2.38)) to be applied in dual variables $\alpha \in \mathbb{R}^m$.

2.4 Support Vector Classification Learning

The methods presented in the last two sections, namely the idea of regularization, and the kernel technique, are elegantly combined in a learning algorithm known as *support vector learning* (SV learning).[16] In the study of SV learning the notion of *margins* is of particular importance. We shall see that the *support vector machine* (SVM) is an implementation of a more general regularization principle known as the *large margin principle*. The greatest drawback of SVMs, that is, the need for zero training error, is resolved by the introduction of *soft margins*. We will demonstrate how both large margin and soft margin algorithms can be viewed in the geometrical picture given in Figure 2.1 on page 23. Finally, we discuss several extensions of the classical SVM algorithm achieved by reparameterization.

2.4.1 Maximizing the Margin

Let us begin by defining what we mean by the margin of a classifier. In Figure 2.6 a training sample z in $\mathbb{R}^2$ together with a classifier (illustrated by the incurred decision surface) is shown. The classifier $f_{\mathbf{w}}$ in Figure 2.6 (a) has a "dead zone" (gray area) separating the two sets of points which is larger than the classifier $f_{\mathbf{w}}$ chosen in Figure 2.6 (b). In both pictures the "dead zone" is the tube around the (linear) decision surface which does not contains any training example $(x_i, y_i) \in z$. To measure the extent of such a tube we can use the norm of the weight vector $\mathbf{w}$ parameterizing the classifier $f_{\mathbf{w}}$. In fact, the size of this tube must be inversely proportional to the minimum real-valued output $y_i \langle \mathbf{x}_i, \mathbf{w} \rangle$ of a classifier $\mathbf{w}$ on a

16 Vapnik also introduced the term *support vector machines* (SVMs) for learning algorithms of the "support vector" type.

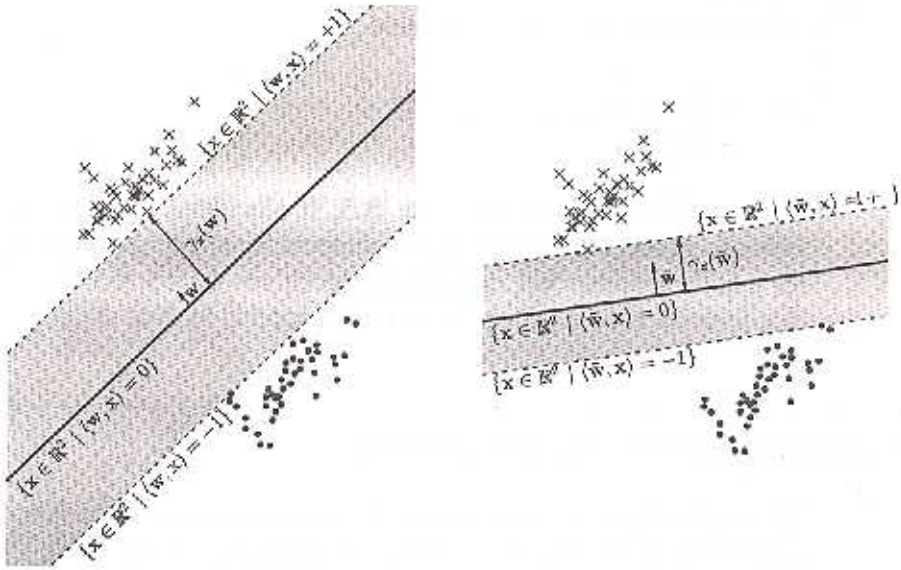

Figure 2.6 Geometrical margins of a plane (thick solid line) in $\mathbb{R}^2$. The crosses ($y_i = +1$) and dots ($y_i = -1$) represent labeled examples $\mathbf{x}_i$. **(Left)** The classifier $f_{\mathbf{w}}$ with the largest geometrical margin $\gamma_z(\mathbf{w})$. Note that this quantity is invariant under rescaling of the weight vector. **(Right)** A classifier $f_{\tilde{\mathbf{w}}}$ with a smaller geometrical margin $\gamma_z(\tilde{\mathbf{w}})$. Since $\min_{(x_i,y_i)\in z} y_i \langle \mathbf{x}_i, \tilde{\mathbf{w}} \rangle = 1$, $\|\tilde{\mathbf{w}}\|$ can be used to measure the extent of the gray zone tube by $\gamma_z(\tilde{\mathbf{w}}) = 1/\|\tilde{\mathbf{w}}\|$.

given training sample z. This quantity is also known as the functional margin on the training sample z and needs to be normalized to be useful for comparison across different weight vectors $\mathbf{w}$ not necessarily of unit length. More precisely, when normalizing the real-valued outputs by the norm of the weight vector $\mathbf{w}$ (which is equivalent to considering the real-valued outputs of normalized weight vectors $\mathbf{w}/\|\mathbf{w}\|$ only) we obtain a confidence measure comparable across different hyperplanes. The following definition introduces the different notions of margins more formally.

Definition 2.30 (Margins) *Suppose we are given a training sample $z = (x, y) \in \mathcal{Z}^m$, a mapping $\phi : \mathcal{X} \to \mathcal{K} \subseteq \ell_2^n$ and a vector $\mathbf{w} \in \mathcal{K}$. For the hyperplane having normal $\mathbf{w}$ we define the*

- functional margin $\tilde{\gamma}_i(\mathbf{w})$ *on an example $(x_i, y_i) \in z$ to be* $\tilde{\gamma}_i(\mathbf{w}) \stackrel{\text{def}}{=} y_i \langle \mathbf{x}_i, \mathbf{w} \rangle$,

- functional margin $\tilde{\gamma}_z(\mathbf{w})$ *on a training sample* z *to be* $\tilde{\gamma}_z(\mathbf{w}) \stackrel{\text{def}}{=} \min_{(x_i,y_i)\in z} \tilde{\gamma}_i(\mathbf{w})$,
- geometrical margin $\gamma_i(\mathbf{w})$ *on an example* $(x_i, y_i) \in z$ *to be* $\gamma_i(\mathbf{w}) \stackrel{\text{def}}{=} \tilde{\gamma}_i(\mathbf{w}) / \|\mathbf{w}\|$,
- geometrical margin $\gamma_z(\mathbf{w})$ *on a training sample* z *to be* $\gamma_z(\mathbf{w}) \stackrel{\text{def}}{=} \tilde{\gamma}_z(\mathbf{w}) / \|\mathbf{w}\|$.

Note that $\tilde{\gamma}_i(\mathbf{w}) > 0$ *implies correct classification of* $(x_i, y_i) \in z$. *Furthermore, for* $\mathbf{w} \in \mathcal{W}$ *the functional and geometrical margin coincide.*

In 1962 Novikoff proved a theorem for perceptrons which was, in 1964, extended to linear classifiers in kernel space. The theorem shows that the number of corrections in the perceptron learning algorithm is provably decreasing for training samples which admit a large margin.

Theorem 2.31 (Perceptron convergence theorem) *Let* $z = (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{Z}^m$ *be a training sample, let* $\boldsymbol{\phi} : \mathcal{X} \to \mathcal{K} \subseteq \ell_2^n$ *be a fixed feature map, and let* $\varsigma = \max_{x_i \in \boldsymbol{x}} \|\boldsymbol{\phi}(x_i)\|$ *be the smallest radius of a sphere enclosing all the mapped training objects* $\boldsymbol{x}$. *Suppose that there exists a vector* $\mathbf{w}^* \in \mathcal{W}$ *such that* $\tilde{\gamma}_z(\mathbf{w}^*) = \gamma_z(\mathbf{w}^*) > 0$. *Then the number of mistakes made by the perceptron learning algorithm on* z *is at most*

$$\left(\frac{\varsigma}{\gamma_z(\mathbf{w}^*)}\right)^2 .$$

The proof is given in Appendix B.4. This theorem answers one of the questions associated with perceptron learning, that is, the number of steps until convergence. The theorem was one of the first theoretical justifications of the idea that large margins yield better classifiers; here in terms of mistakes during learning. We shall see in Part II that large margins indeed yield better classifiers in terms of expected risk.

Let $\mathcal{F}$ and $\mathcal{K}$ be the RKHS and feature space connected with the Mercer kernel k, respectively. The classifier $\mathbf{w}$ with the largest margin $\gamma_z(\mathbf{w})$ on a given training sample can be written as

$$\mathbf{w}_{\text{SVM}} \stackrel{\text{def}}{=} \operatorname*{argmax}_{\mathbf{w}\in\mathcal{W}} \gamma_z(\mathbf{w}) = \operatorname*{argmax}_{\mathbf{w}\in\mathcal{K}} \frac{1}{\|\mathbf{w}\|}\tilde{\gamma}_z(\mathbf{w}) . \tag{2.40}$$

Two methods of casting the problem of finding this classifier into a regularization framework are conceivable. One method is to refine the (coarse) l_{0-1} loss function given in equation (2.9) by exploiting the minimum real-valued output $\gamma_z(\mathbf{w})$ of

each classifier $\mathbf{w} \in \mathcal{W}$. A second option is to fix the minimum real-valued output $\tilde{\gamma}_z(\mathbf{w})$ of the classifier $\mathbf{w} \in \mathcal{K}$ and to use the norm $\|\mathbf{w}\|$ of each classifier to measure its complexity. Though the latter is better known in the SV community we shall present both formulations.

1. Fix the norm of the classifiers to unity (as done in Novikoff's theorem), then we must maximize the geometrical margin. More formally, in terms of equation (2.38) we have

$$\mathbf{w}_{\mathrm{SVM}} = \operatorname*{argmin}_{\mathbf{w} \in \mathcal{W}} \; l_{\mathrm{margin}}(\gamma_z(\mathbf{w})) \;, \tag{2.41}$$

where

$$l_{\mathrm{margin}}(t) \stackrel{\mathrm{def}}{=} -t \;. \tag{2.42}$$

A more convenient notation of this minimization problem is

$$\begin{array}{ll} \text{maximize} & \min(f_{\mathbf{w}}(x_1), \ldots, f_{\mathbf{w}}(x_m)) = \gamma_z(\mathbf{w}) \\ \text{subject to} & \|f_{\mathbf{w}}\|^2 = \|\mathbf{w}\|^2 = 1 \;. \end{array}$$

This optimization problem has several difficulties associated with it. First, the objective function is neither linear nor quadratic. Further, the constraints are nonlinear. Hence, from an algorithmic viewpoint this optimization problem is difficult to solve. Nonetheless, due to the independence of the hypothesis space from the training sample it is very useful in the study of the generalization error.

2. Fix the functional margin to unity and minimize the norm $\|\mathbf{w}\|$ of the weight vector. More formally, the set of all classifiers considered for learning is

$$\mathcal{W}(z) \stackrel{\mathrm{def}}{=} \{\mathbf{w} \in \mathcal{K} \mid \tilde{\gamma}_z(\mathbf{w}) = 1\} \;, \tag{2.43}$$

which are known as *canonical hyperplanes*. Clearly, this definition of the hypothesis space is data dependent which makes a theoretical analysis quite intricate[17]. The advantage of this formulation becomes apparent if we consider the corresponding risk functional:

$$\mathbf{w}_{\mathrm{SVM}} \propto \operatorname*{argmin}_{\mathbf{w} \in \mathcal{W}(z)} \|f_{\mathbf{w}}\|^2 = \operatorname*{argmin}_{\mathbf{w} \in \mathcal{W}(z)} \|\mathbf{w}\|^2 \;. \tag{2.44}$$

17 In general, the hypothesis space must be independent of the training sample. The training sample dependence on the hypothesis space for Mercer kernels is resolved in Theorem 2.29. Note, however, that this theorem does not apply to canonical hyperplanes.

The risk functional seems to imply that we minimize a complexity or structural risk, but this is wrong. In fact, the lack of any empirical term in the risk functional is merely due to the formulation which uses a data dependent hypothesis space (2.43). If we cast the minimization of this risk functional in a convex programming framework we obtain

$$\begin{array}{lll} \text{minimize} & \|\mathbf{w}\|^2 = \|f_{\mathbf{w}}\|^2 & \\ \text{subject to} & y_i\,(\mathbf{x}_i, \mathbf{w}) \;\geq\; 1 & i = 1, \ldots, m\,. \end{array} \tag{2.45}$$

This optimization problem is much more computationally amenable. Here, the objective function is quadratic and the constraints are linear. As a consequence, the solution must be expressible in its dual form. Introducing m Lagrangian multipliers α_i for the linear constraints (which turn out to be the expansion coefficients of the weight vector $\mathbf{w}$ in terms of the mapped training objects), taking the derivative w.r.t. $\mathbf{w}$ and back-inserting into the Lagrangian, we obtain the following *Wolfe dual* (for details see Section B.5)

$$W\,(\boldsymbol{\alpha}) = \boldsymbol{\alpha}'\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}'\mathbf{Y}\mathbf{G}\mathbf{Y}\boldsymbol{\alpha}\,, \tag{2.46}$$

which needs to be maximized in the positive quadrant $\mathbf{0} \leq \boldsymbol{\alpha}$,

$$\hat{\boldsymbol{\alpha}} = \underset{\mathbf{0}\leq\boldsymbol{\alpha}}{\operatorname{argmax}}\; W\,(\boldsymbol{\alpha})\;.$$

Here, $\mathbf{G}$ is the $m \times m$ Gram matrix defined by equation (2.18) and $\mathbf{Y} \stackrel{\text{def}}{=} \operatorname{diag}\,(y_1, \ldots, y_m)$. Note, however, that the solution

$$\mathbf{w}_{\text{SVM}} = \sum_{i=1}^{m} \hat{\alpha}_i\, y_i\, \mathbf{x}_i$$

is equivalent to the solution of optimization problem (2.41) up to a scaling factor. Using decomposition techniques to solve the problem, the computational effort is roughly of order $\mathcal{O}\,(m^2)$.

2.4.2 Soft Margins—Learning with Training Error

The algorithm presented in the last subsection is clearly restricted to training samples which are linearly separable. One way to deal with this insufficiency is to use "powerful" kernels (like an RBF kernel with very small σ) which makes each training sample separable in feature space. Although this would not cause

any computational difficulties, the "large expressive" power of the classifiers in feature space may lead to overfitting, that is, a large discrepancy between empirical risk (which was previously zero) and true risk of a classifier. Moreover, the above algorithm is "nonrobust" in the sense that one outlier (a training point $(x_i, y_i) \in z$ whose removal would lead to a large increase in margin) can cause the learning algorithm to converge very slowly or, even worse, make it impossible to apply at all (if $\gamma_i(\mathbf{w}) < 0$ for all $\mathbf{w} \in \mathcal{W}$).

In order to overcome this insufficiency we introduce a heuristic which has become known as the *soft margin SVM*. The idea exploited is to upper bound the zero-one loss l_{0-1} as given in equation (2.9) by a linear or quadratic function (see Figure 2.7),

$$\begin{aligned} l_{0-1}(f(x), y) = \mathbf{1}_{-yf(x)>0} &\leq \max\{1 - yf(x), 0\} = l_{\text{lin}}(f(x), y), \\ l_{0-1}(f(x), y) = \mathbf{1}_{-yf(x)>0} &\leq \max\{1 - yf(x), 0\}^2 = l_{\text{quad}}(f(x), y). \end{aligned} \tag{2.47}$$

It is worth mentioning that, due to the cut off at a real-valued output of one (on the correct side of the decision surface), the norm $\|f\|$ can still serve as a regularizer. Viewed this way, the idea is in the spirit of the second parameterization of the optimization problem of large margins (see equation (2.40)).

Linear Approximation

Let us consider the case of a linear approximation. Given a tradeoff parameter $\lambda > 0$, the regularization functional becomes

$$R_{\text{reg}}[f_{\mathbf{w}}, z] = \frac{1}{m}\sum_{i=1}^{m} l_{\text{lin}}(f_{\mathbf{w}}(x_i), y_i) + \lambda \|f_{\mathbf{w}}\|^2,$$

or equivalently

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{m} \xi_i + \lambda m \|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i(\mathbf{x}_i, \mathbf{w}) \geq 1 - \xi_i \qquad i = 1, \ldots, m, \\ & \boldsymbol{\xi} \geq \mathbf{0}. \end{aligned} \tag{2.48}$$

Transforming this into an optimization problem involving the corresponding Wolfe dual we must maximize an equation of the form (2.46), but this time in the "box" $\mathbf{0} \leq \boldsymbol{\alpha} \leq \frac{1}{2\lambda m}\mathbf{1}$ (see Section B.5). In the limit $\lambda \to 0$ we obtain the "hard margin" SVM because there is no upper bound on $\boldsymbol{\alpha}$. Another explanation of

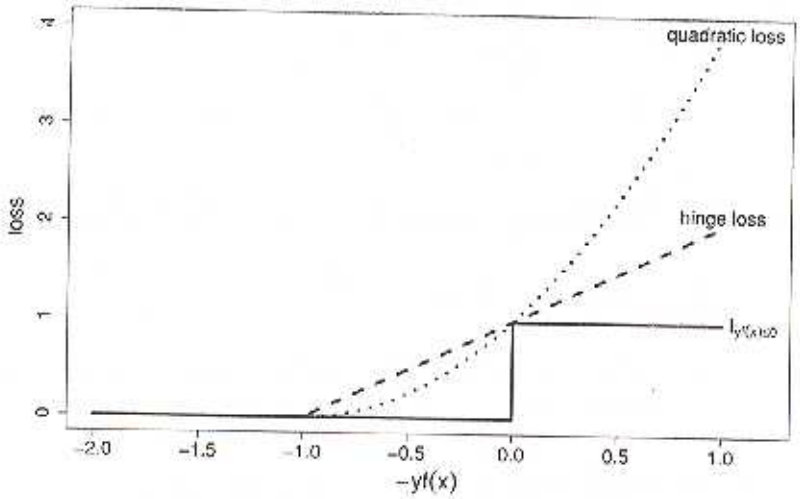

Figure 2.7 Approximation to the Heaviside step function $\mathbf{I}_{yf(x)\leq 0}$ (solid line) by the so-called "hinge loss" (dashed line) and a quadratic margin loss (dotted line). The $x-$axis contains the negative real-valued output $-yf(\mathbf{x})$ which is positive in the case of misclassification of x by f.

this equivalence is given by the fact that the objective function is proportional to $\frac{1}{\lambda m}\sum_{i=1}^{m}\xi_i + \|\mathbf{w}\|^2$. Thus, in the limit of $\lambda \to 0$, any $\mathbf{w}$ for which $\boldsymbol{\xi} \neq \mathbf{0}$ incurs an infinitely large value of the objective function and therefore in the optimum $\sum_{i=1}^{m}\xi_i = 0$. Note that by virtue of this formulation the "box" is decreased with increasing training sample size.

Quadratic Approximation

Though not as popular in the SV community, the quadratic approximation has proven to be successful in real world applications. Formally, the regularization functional becomes

$$R_{\mathrm{reg}}[f_{\mathbf{w}}, z] = \frac{1}{m}\sum_{i=1}^{m} l_{\mathrm{quad}}(f_{\mathbf{w}}(x_i), y_i) + \lambda \|f_{\mathbf{w}}\|^2 ,$$

which in its equivalent form is

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{m} \xi_i^2 + \lambda m \|\mathbf{w}\|^2 \\ \text{subject to} & y_i (\mathbf{x}_i, \mathbf{w}) \geq 1 - \xi_i \qquad i = 1, \ldots, m, \\ & \boldsymbol{\xi} \geq \mathbf{0}. \end{array} \tag{2.49}$$

The corresponding Wolfe dual (derived in Section B.5) is given by

$$W(\boldsymbol{\alpha}) = \boldsymbol{\alpha}'\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}'\mathbf{Y}\mathbf{G}\mathbf{Y}\boldsymbol{\alpha} - \frac{\lambda m}{2}\boldsymbol{\alpha}'\boldsymbol{\alpha},$$

and must be maximized in the positive quadrant $\mathbf{0} \leq \boldsymbol{\alpha}$. This can equivalently be expressed by a change of the Gram matrix, i.e.,

$$W(\boldsymbol{\alpha}) = \boldsymbol{\alpha}'\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}'\mathbf{Y}\widetilde{\mathbf{G}}\mathbf{Y}\boldsymbol{\alpha}, \qquad \widetilde{\mathbf{G}} = \mathbf{G} + \lambda m \mathbf{I}. \tag{2.50}$$

Remark 2.32 (Data independent hypothesis spaces) *The two algorithms presented in this subsection use the idea of fixing the functional margin to unity. This allows the geometrical margin to be controlled by the norm $\|\mathbf{w}\|$ of the weight vector $\mathbf{w}$. As we have seen in the previous subsection there also exists a "data independent" formulation. In the case of a quadratic soft margin loss the formulation is apparent from the change of the Gram matrix: The quadratic soft margin SVM is equivalent to a hard margin SVM if we change the Gram matrix $\mathbf{G}$ to $\mathbf{G} + \lambda m \mathbf{I}$. Furthermore, in the hard margin case, we could alternatively have the hypothesis space being the unit hypersphere in feature space. As a consequence thereof, all we need to consider is the change in the feature space, if we* penalize the diagonal of the Gram matrix *by λm.*

Remark 2.33 (Cost matrices) *In Example 2.7 we showed how different a-priori class probabilities $\mathbf{P}_Y(-1)$ and $\mathbf{P}_Y(+1)$ can be incorporated through the use of a cost matrix loss function. In the case of soft margin loss this can be approximately achieved by using different values $\lambda_+ \in \mathbb{R}^+$ and $\lambda_- \in \mathbb{R}^+$ at the constraints for the training points of class $+1$ and -1, respectively. As the (general) regularizer is inversely related to the allowed violation of constraints it follows that the underrepresented class having smaller prior probability should have the larger λ value.*

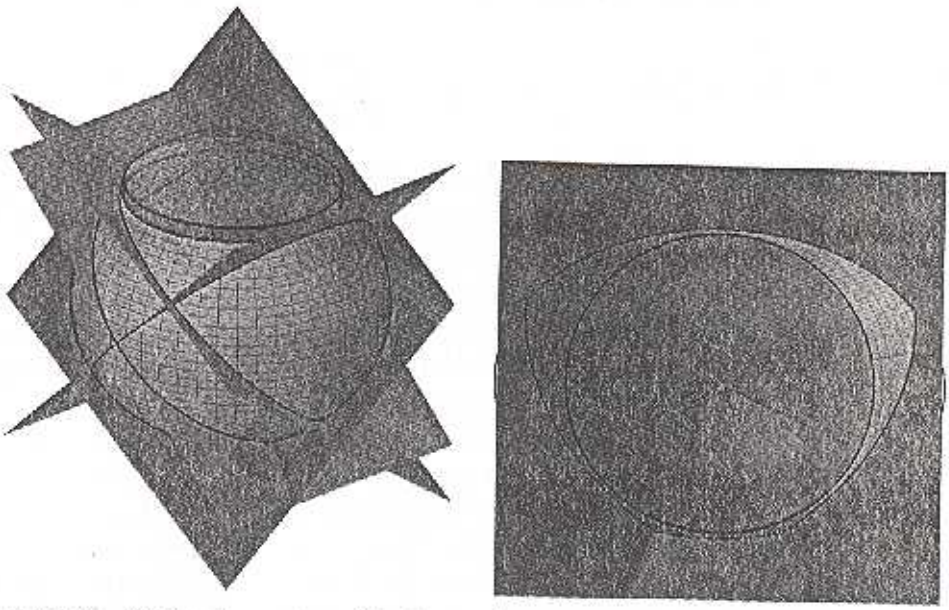

Figure 2.8 Finding the center of the largest inscribable ball in version space. **(Left)** In this example four training points were given which incur the depicted four planes. Let us assume that the labeling of the training sample was such that the polyhedra on top of the sphere is the version space. Then, the SV learning algorithm finds the (weight) vector $\mathbf{w}$ on top of the sphere as the center of the largest inscribable ball $\mathcal{B}_1(\mathbf{w})$ (transparent cap). Here, we assumed $\|y_i \mathbf{x}_i\| = \|\mathbf{x}_i\|$ to be constant. The distance of the $\mathbf{w}$ from the hyperplanes (dark line) is proportional to the margin $\gamma_z(\mathbf{w})$ (see text). **(Right)** Viewed from top we see that the version space $V(z)$ is a bended convex body into which we can fully inscribe a circle of radius proportional to $\gamma_z(\mathbf{w})$.

2.4.3 Geometrical Viewpoints on Margin Maximization

In the previous two subsections the SV learning algorithms were introduced purely from a margin maximization perspective. In order to associate these algorithms with the geometrical picture given in Figure 2.1 on page 23 we note that, for a fixed point $(x_i, y_i) \in z$, the geometrical margin $\gamma_i(\tilde{\mathbf{w}})$ can be read as the distance of the linear classifier having normal $\tilde{\mathbf{w}}$ to the hyperplane $\{\mathbf{w} \in \mathcal{K} \mid y_i \langle \mathbf{x}_i, \mathbf{w} \rangle = 0\}$. In fact, the Euclidean distance of the point $\tilde{\mathbf{w}}$ from the hyperplane having normal $y_i \mathbf{x}_i$ is $y_i \langle \mathbf{x}_i, \tilde{\mathbf{w}} \rangle / \|y_i \mathbf{x}_i\| = \gamma_i(\tilde{\mathbf{w}}) / \|\mathbf{x}_i\|$. For the moment let us assume that $\|\mathbf{x}_i\|$ is constant for all x_i in the training objects $x \in \mathcal{X}^m$. Then, if a classifier $f_{\tilde{\mathbf{w}}}$ achieves

a margin of $\gamma_z(\tilde{\mathbf{w}})$ on the training sample z we know that the ball,

$$\mathcal{B}_\tau(\tilde{\mathbf{w}}) = \left\{ \mathbf{w} \in \mathcal{W} \;\middle|\; \|\mathbf{w} - \tilde{\mathbf{w}}\| < \frac{\gamma_z(\tilde{\mathbf{w}})}{\|\mathbf{x}_i\|} \right\} \subset V(z)$$

of radius $\tau = \gamma_z(\tilde{\mathbf{w}}) / \|\mathbf{x}_i\|$ is totally inscribable in version space $V(z)$. Henceforth, maximizing $\gamma_z(\tilde{\mathbf{w}})$ is equivalent to finding the center of the largest inscribable ball in version space (see Figure 2.8).

The situation changes if we drop the assumption that $\|\mathbf{x}_i\|$ is constant. In this case, training objects for which $\|\mathbf{x}_i\|$ is very large effectively minimize the radius τ of the largest inscribable ball. If we consider the center of the largest inscribable ball as an approximation to the center of mass of version space $V(z)$ (see also Section 3.4) we see that normalizing the $\mathbf{x}_i$'s to unit length is crucial to finding a good approximation for this point.

The geometrical intuition still holds if we consider the quadratic approximation presented in Subsection 2.4.2. The effect of the diagonal penalization is to add a new basis axis for each training point $(x_i, y_i) \in z$. Hence, in this new space the quadratic SVM tries to find the center of the largest inscribable ball. Needless to say that we again assume the $\mathbf{x}_i$'s to be of constant length $\|\mathbf{x}_i\|$. We shall see in Section 5.1 that the margin $\gamma_z(\tilde{\mathbf{w}})$ is too coarse a measure to be used for bounds on the expected risk if $\|\mathbf{x}_i\| \not\equiv \text{const.}$—especially if we apply the kernel technique.

2.4.4 The ν–Trick and Other Variants

The SV algorithms presented so far constitute the basis of the standard SV tool box. There exist, however, several (heuristic) extensions for the case of multiple classes ($2 < |\mathcal{Y}| < \infty$), regression estimation ($\mathcal{Y} = \mathbb{R}$) and reparameterizations in terms of the assumed noise level $\mathbf{E}_X\left[1 - \max_{y \in \mathcal{Y}}\left(\mathbf{P}_{Y|X=x}(y)\right)\right]$ which we present here.

Multiclass Support Vector Machines

In order to extend the SV learning algorithm to $K = |\mathcal{Y}| > 2$ classes two different strategies have been suggested.

1. The first method is to learn K SV classifiers f_j by labeling all training points having $y_i = j$ with $+1$ and $y_i \neq j$ with -1 during the training of the jth classifier.

In the test stage, the final decision is obtained by

$$f_{\text{multiple}}(x) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \; f_y(x) \; .$$

Clearly, this method learns one classifier for each of the K classes against all the other classes and is hence known as the *one-versus-rest* (o-v-r) method. It can be shown that it is possible to solve the K optimization problems at once. Note that the computational effort is of order $\mathcal{O}\left(Km^2\right)$.

2. The second method is to learn $K(K-1)/2$ SV classifiers. If $1 \leq i < j \leq K$ the classifiers $f_{i,j}$ is learned using only the training samples from the class i and j, labeling them $+1$ and -1, respectively. This method has become known as the *one-versus-one* (o-v-o) method. Given a new test object $x \in \mathcal{X}$, the frequency n_i of "wins" for class i is computed by applying $f_{i,j}$ for all j. This results in a vector $\mathbf{n} = (n_1; \ldots; n_K)$ of frequencies of "wins" of each class. The final decision is made for the most frequent class, i.e.,

$$f_{\text{multiple}}(x) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \; n_y \; .$$

Using a probabilistic model for the frequencies $\mathbf{n}$, different prior probabilities of the classes $y \in \mathcal{Y}$ can be incorporated, resulting in better generalization ability. Instead of solving $K(K-1)/2$ separate optimization problems, it is again possible to combine them in a single optimization problem. If the prior probabilities $\mathbf{P}_Y(j)$ for the K classes are roughly $\frac{1}{K}$, the method scales as $\mathcal{O}\left(m^2\right)$ and is independent of the number of classes.

Recently, a different method for combining the single pairwise decisions has been suggested. By specifying a *directed acyclic graph* (DAG) of consecutive pairwise classifications, it is possible to introduce a class hierarchy. The leaves of such a DAG contain the final decisions which are obtained by exclusion rather than by voting. This method compares favorably with the o-v-o and o-v-r methods.

Support Vector Regression Estimation

In the regression estimation problem we are given a sample of m real target values $\boldsymbol{t} = (t_1, \ldots, t_m) \in \mathbb{R}^m$, rather than m class labels $\boldsymbol{y} = (y_1, \ldots, y_m) \in \mathcal{Y}^m$. In order to extend the SV learning algorithm to this task, we note that an "inversion" of the linear loss l_{lin} suffices in order to use the SV machinery for real-valued outputs t_i. In classification the linear loss $l_{\text{lin}}(f(x), \cdot)$ adds to the total cost, if the real-valued

output of $|f(x)|$ is smaller than 1. For regression estimation it is desirable to have the opposite true, i.e., incurred costs result if $|t - f(x)|$ is very large instead of small. This requirement is formally captured by the *ε–insensitive* loss

$$l_\varepsilon(f(x), t) = \begin{cases} 0 & \text{if } |t - f(x)| \leq \varepsilon \\ |t - f(x)| - \varepsilon & \text{if } |t - f(x)| > \varepsilon \end{cases}. \tag{2.51}$$

Then, one obtains a quadratic programming problem similar to (2.46), this time in $2m$ dual variables α_i and $\tilde{\alpha}_i$—two corresponding to each training point constraint. This is simply due to the fact that f can fail to attain a deviation less than ε on both sides of the given real-valued output t_i, i.e., $t_i - \varepsilon$ and $t_i + \varepsilon$. An appealing feature of this loss is that it leads to sparse solutions, i.e., only a few of the α_i (or $\tilde{\alpha}_i$) are non-zero. For further references that cover the regression estimation problem the interested reader is referred to Section 2.6.

ν–Support Vector Machines for Classification

A major drawback of the soft margin SV learning algorithm given in the form (2.48) is the lack of control over how many training points will be considered as margin errors or "outliers", that is, how many have $\tilde{\gamma}_i(\mathbf{w}_{\mathrm{SVM}}) < 1$. This is essentially due to the fact that we fixed the functional margin to one. By a simple reparameterization it is possible to make the functional margin itself a variable of the optimization problem. One can show that the solution of the following optimization problem has the property that the new parameter ν bounds the fraction of *margin errors* $\frac{1}{m}|\{(x_i, y_i) \in z \mid \tilde{\gamma}_i(\mathbf{w}_{\mathrm{SVM}}) < \rho\}|$ from above:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{m}\sum_{i=1}^{m}\xi_i - \nu\rho + \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i\langle \mathbf{x}_i, \mathbf{w}\rangle \geq \rho - \xi_i \qquad i = 1, \ldots, m, \\ & \boldsymbol{\xi} \geq \mathbf{0}, \ \rho \geq 0. \end{aligned} \tag{2.52}$$

It can be shown that, for each value of $\nu \in [0, 1]$, there exists a value of $\lambda \in \mathbb{R}^+$ such that the solution $\mathbf{w}_\nu$ and $\mathbf{w}_\lambda$ found by solving (2.52) and (2.48) have the same geometrical margins $\gamma_z(\mathbf{w}_\nu) = \gamma_z(\mathbf{w}_\lambda)$. Thus we could try different values of λ in the standard linear soft margin SVM to obtain a required fraction of margin errors. The appealing property of the problem (2.52) is that this adjustment is done within the one optimization problem (see Section B.5). Another property which can be proved is that, for all probability models where neither $\mathbf{P}_X(\{X, 1\})$ nor

$\mathsf{P}_X(\{X, -1\})$ contains any discrete component, ν asymptotically equals the fraction of margin errors. Hence, we can incorporate prior knowledge of the noise level $\mathsf{E}_X\left[1 - \max_{y \in \mathcal{Y}}\left(\mathsf{P}_{Y|X=x}(y)\right)\right]$ via ν. Excluding all training points for which the real-valued output is less than ρ in absolute value, the geometrical margin of the solution on the remaining training points is $\rho / \|\mathbf{w}\|$.

2.5 Adaptive Margin Machines

In this last section we will introduce an algorithm which is based on a conceptually different principle. Our new approach is motivated by a recently derived leave-one-out bound on the generalization error of kernel classifiers. Let us start by introducing the concept of the leave-one-out error.

2.5.1 Assessment of Learning Algorithms

Whilst the mathematical model of learning to be introduced in Part II of this book gives some motivation for the algorithms introduced so far, the derived bounds are often too loose to be useful in practical applications. A completely different approach can be taken if we study the expected risk of *a learning algorithm* $\mathcal{A}$ rather than any hypothesis.

Definition 2.34 (Expected risk of a learning algorithm) *Given an algorithm* $\mathcal{A} : \bigcup_{m=1}^{\infty} \mathcal{Z}^m \to \mathcal{F}$, *a loss function* $l : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$ *and a training sample size* $m \in \mathbb{N}$, *the* expected risk $R[\mathcal{A}, m]$ of the learning algorithm $\mathcal{A}$ *is defined by*

$$R[\mathcal{A}, m] \stackrel{\text{def}}{=} \mathsf{E}_{Z^m}\left[R\left[\mathcal{A}(\mathbf{Z})\right]\right].$$

Note that this quantity does not bound the expected risk of the *one* classifier learned from a training sample z but the *average expected risk performance* of the algorithm $\mathcal{A}$. For any training sample z, an almost unbiased estimator of this quantity is given by the *leave-one-out error* $R_{\text{loo}}[\mathcal{A}, z]$ of $\mathcal{A}$.

Definition 2.35 (Leave-one-out error) *Given an algorithm* $\mathcal{A} : \bigcup_{m=1}^{\infty} \mathcal{Z}^m \to \mathcal{F}$, *a loss function* $l : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$ *and a training sample* $z \in \mathcal{Z}^m$, *the* leave-one-out

error *is defined by*

$$R_{\mathrm{loo}}[\mathcal{A}, z] \stackrel{\mathrm{def}}{=} \frac{1}{m} \sum_{i=1}^{m} l\left(\mathcal{A}\left(\left(z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_m\right)\right)\left(x_i\right), y_i\right) .$$

This measure counts the fraction of examples that are misclassified if we leave them out for learning when using the algorithm $\mathcal{A}$. The unbiasedness of the estimator is made more precise in the following proposition.

Theorem 2.36 (Unbiasedness of the leave-one-out error) *Given a fixed measure* $\mathbf{P}_Z$*, a fixed hypothesis space* $\mathcal{F}$*, a fixed loss* l *and a fixed learning algorithm* $\mathcal{A} : \bigcup_{m=1}^{\infty} \mathcal{Z}^m \to \mathcal{F}$*, the leave-one-out error is almost unbiased, that is,*

$$\mathbf{E}_{\mathbf{Z}^m}\left[R_{\mathrm{loo}}[\mathcal{A}, \mathbf{Z}]\right] = R[\mathcal{A}, m-1] .$$

Proof In order to prove the result we note that

$$\begin{aligned}
\mathbf{E}_{\mathbf{Z}^m}\left[R_{\mathrm{loo}}[\mathcal{A}, \mathbf{Z}]\right] &= \mathbf{E}_{\mathbf{Z}^m}\left[\frac{1}{m} \sum_{i=1}^{m} l\left(\mathcal{A}\left(\left(\mathbf{Z}_1, \ldots, \mathbf{Z}_{i-1}, \mathbf{Z}_{i+1}, \ldots, \mathbf{Z}_m\right)\right)\left(\mathbf{X}_i\right), \mathbf{Y}_i\right)\right] \\
&= \frac{1}{m} \sum_{i=1}^{m} \mathbf{E}_{\mathbf{Z}^m}\left[l\left(\mathcal{A}\left(\left(\mathbf{Z}_1, \ldots, \mathbf{Z}_{i-1}, \mathbf{Z}_{i+1}, \ldots, \mathbf{Z}_m\right)\right)\left(\mathbf{X}_i\right), \mathbf{Y}_i\right)\right] \\
&= \frac{1}{m} \sum_{i=1}^{m} \mathbf{E}_{\mathbf{Z}^{m-1}}\left[\mathbf{E}_{\mathbf{XY}|\mathbf{Z}^{m-1}=z}\left[l\left(\mathcal{A}\left(z\right)\left(\mathbf{X}\right), \mathbf{Y}\right)\right]\right] \\
&= \mathbf{E}_{\mathbf{Z}^{m-1}}\left[R\left[\mathcal{A}\left(\mathbf{Z}\right)\right]\right] = R[\mathcal{A}, m-1] .
\end{aligned}$$

The theorem is proved. ■

Despite the fact that this result allows us to obtain a precise estimate of the expected risk of the learning algorithm, its computation is very time consuming as the learning algorithm must be invoked m times. Therefore, it is desirable to have a bound on this quantity which can be computed solely on the basis of the training sample z and the learned hypothesis $\mathcal{A}(z)$. As demonstrated in Section 2.4, a rather powerful class of learning algorithms is given by

$$\hat{\alpha} = \operatorname*{argmax}_{0 \le \alpha \le u} W(\alpha)$$

$$W(\boldsymbol{\alpha}) = -\frac{1}{2}\boldsymbol{\alpha}'\mathbf{YGY}\boldsymbol{\alpha} + \sum_{i=1}^{m} J(\alpha_i), \tag{2.53}$$

where $J : \mathbb{R} \to \mathbb{R}$ is a fixed function, $\mathbf{u}$ is an $m \times 1$ vector of positive real numbers, $\mathbf{Y} \stackrel{\text{def}}{=} \text{diag}(y_1, \ldots, y_m)$ and $\mathbf{G}$ is the $m \times m$ Gram matrix given by equation (2.18). Based on the vector $\hat{\boldsymbol{\alpha}} \in \mathbb{R}^m$, the linear classifier f is then given by

$$f(x) = \langle \hat{\mathbf{w}}, \mathbf{x} \rangle = \sum_{i=1}^{m} \hat{\alpha}_i y_i k(x_i, x) \quad \Leftrightarrow \quad \hat{\mathbf{w}} = \sum_{i=1}^{m} \hat{\alpha}_i y_i \mathbf{x}_i . \tag{2.54}$$

We can give the following bound on the leave-one-out error $R_{\text{loo}}[\mathcal{A}_W, z]$.

Theorem 2.37 (Leave-One-Out Bound) *Suppose we are given a training sample $z \in \mathcal{Z}^m$ and a Mercer kernel k. Let $\hat{\boldsymbol{\alpha}}$ be the maximizing coefficients of (2.53). Then an upper bound on the leave-one-out error of $\mathcal{A}_W$ is given by*

$$R_{\text{loo}}[\mathcal{A}_W, z] \leq \frac{1}{m} \sum_{i=1}^{m} \Theta\left(-y_i \sum_{\substack{j=1 \\ j \neq i}}^{m} \hat{\alpha}_j y_j k(x_i, x_j) \right), \tag{2.55}$$

where $\Theta(t) = \mathbf{I}_{t \geq 0}$ is the Heaviside step function.

The proof is given in Appendix B.6. For support vector machines V. Vapnik has shown that the leave-one-out error is bounded by the ratio of the number of non-zero coefficients $\hat{\alpha}_i$ to the number m of training examples. The bound given in Theorem 2.37 is slightly tighter than Vapnik's leave-one-out bound. This is easy to see because all training points that have $\hat{\alpha}_i = 0$ cannot be leave-one-out errors in either bound. Vapnik's bound assumes all support vectors (all training points with $\hat{\alpha}_i > 0$) are leave-one-out errors, whereas they only contribute as errors in equation (2.55) if $y_i \sum_{\substack{j=1 \\ j \neq i}}^{m} \hat{\alpha}_j y_j k(x_i, x_j) \leq 0$. In practice this means that the bound (2.55) is tighter for less sparse solutions.

2.5.2 Leave-One-Out Machines

Theorem 2.37 suggests an algorithm which directly minimizes the expression in the bound. The difficulty is that the resulting objective function will contain the step function $\mathbf{I}_{t \geq 0}$. The idea we exploit is similar to the idea of soft-margins in

SVMs, where the step function is upper bounded by a piecewise linear function, also known as the hinge loss (see Figure 2.7). Hence, introducing slack variables, gives the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{m} \xi_i \\ \text{subject to} \quad & y_i \sum_{\substack{j=1 \\ j \neq i}}^{m} \alpha_j y_j k\left(x_i, x_j\right) \geq 1 - \xi_i \qquad i = 1, \ldots, m, \\ & \alpha \geq \mathbf{0}, \xi \geq \mathbf{0}. \end{aligned} \tag{2.56}$$

For further classification of new test objects we use the decision rule given in equation (2.54). Let us study the resulting method which we call a *leave-one-out machine* (LOOM).

First, the technique appears to have no free regularization parameter. This should be compared with support vector machines, which control the amount of regularization through the free parameter λ. For SVMs, in the case of $\lambda \to 0$ one obtains a hard margin classifier with no training errors. In the case of linearly inseparable datasets in feature space (through noise, outliers or class overlap) one must admit some training errors (by constructing soft margins). To find the best choice of training error/margin tradeoff one must choose the appropriate value of λ. In leave-one-out machines a soft margin is automatically constructed. This happens because the algorithm does not attempt to minimize the number of training errors—it minimizes the number of training points that are classified incorrectly even when they are removed from the linear combination which forms the decision rule. However, if one can classify a training point correctly when it is removed from the linear combination, then it will always be classified correctly when it is placed back into the rule. This can be seen as $\alpha_i y_i k\left(x_i, x_i\right)$ always has the same sign as y_i; any training point is pushed further from the decision boundary by its own component of the linear combination. Note also that summing for all $j \neq i$ in the constraint (2.56) is equivalent to setting the diagonal of the Gram matrix $\mathbf{G}$ to zero and instead summing for all j. Thus, the regularization employed by leave-one-out machines disregards the values $k\left(x_i, x_i\right)$ for all i.

Second, as for support vector machines, the solutions $\hat{\alpha} \in \mathbb{R}^m$ can be sparse in terms of the expansion vector; that is, only some of the coefficients $\hat{\alpha}_i$ are non-zero. As the coefficient of a training point does not contribute to its leave-one-out error in constraint (2.56), the algorithm does not assign a non-zero value to the

coefficient of a training point in order to correctly classify it. A training point has to be classified correctly by the training points of the same label that are close to it, but the point itself makes no contribution to its own classification in training.

2.5.3 Pitfalls of Minimizing a Leave-One-Out Bound

The core idea of the presented algorithm is to directly minimize the leave-one-out bound. Thus, it seems that we are able to control the generalization ability of an algorithm disregarding quantities like the margin. This is not true in general[18] and in particular the presented algorithm is not able to achieve this goal. There are some pitfalls associated with minimizing a leave-one-out bound:

1. In order to get a bound on the leave-one-out error we must specify the algorithm $\mathcal{A}$ *beforehand*. This is often done by specifying the form of the objective function which is to be maximized (or minimized) during learning. In our particular case we see that Theorem 2.37 only considers algorithms defined by the maximization of $W(\boldsymbol{\alpha})$ with the "box" constraint $\mathbf{0} \leq \boldsymbol{\alpha} \leq \mathbf{u}$. By changing the learning algorithm to minimize the bound itself we may well develop an optimization algorithm which is no longer compatible with the assumptions of the theorem. This is true in particular for leave-one-out machines which are no longer in the class of algorithms considered by Theorem 2.37—whose bound they are aimed at minimizing. Further, instead of minimizing the bound directly we are using the hinge loss as an upper bound on the Heaviside step function.

2. The leave-one-out bound does not provide any guarantee about the generalization error $R[\mathcal{A}, z]$ (see Definition 2.10). Nonetheless, if the leave-one-out error is small then we know that, for most training samples $z \in \mathcal{Z}^m$, the resulting classifier has to have an expected risk close to that given by the bound. This is due to Hoeffding's bound which says that for bounded loss (the expected risk of a hypothesis f is bounded to the interval $[0, 1]$) the expected risk $R[\mathcal{A}(z)]$ of the learned classifier $\mathcal{A}(z)$ is close to the expectation of the expected risk (bounded by the leave-one-out bound) with high probability over the random choice of the training sample.[19] Note, however, that the leave-one-out estimate does not provide any information about the variance of the expected risk. Such information would allow the application of tighter bounds, for example, Chebyshev's bound.

---

18 Part II, Section 4.3, shows that there are models of learning which allow an algorithm to *directly* minimize a *bound* on its generalization error. This should not be confused with the possibility of controlling the generalization error of the algorithm itself.

19 We shall exploit this idea further in Part II, Section 5.3.

3. The original motivation behind the use of the leave-one-out error was to measure the goodness of the hypothesis space $\mathcal{F}$ and of the learning algorithm $\mathcal{A}$ for the learning problem given by the unknown probability measure $\mathbf{P_Z}$. Commonly, the leave-one-out error is used to select among different models $\mathcal{F}_1, \mathcal{F}_2, \ldots$ for a given learning algorithm $\mathcal{A}$. In this sense, minimizing the leave-one-out error is more a model selection strategy than a learning paradigm within a fixed model.

Definition 2.38 (Model selection) *Suppose we are given $r \in \mathbb{N}$ fixed learning algorithms $\mathcal{A}_i : \cup_{m=1}^{\infty} \mathcal{Z}^m \to \mathcal{Y}^{\mathcal{X}}$ which map training samples z to classifiers $h \in \mathcal{Y}^{\mathcal{X}}$. Then, given a training sample $z \in \mathcal{Z}^m$, the problem of* model selection *is to identify the learning algorithm $\mathcal{A}_i$ which would lead to a classifier $\mathcal{A}_i(z)$ possessing the smallest expected risk, i.e., find the algorithm $\mathcal{A}_z$ such that*

$$\mathcal{A}_z = \operatorname*{argmin}_{\mathcal{A}_i} R\left[\mathcal{A}_i(z)\right].$$

If we have a fixed learning procedure $\mathcal{A}_\chi : \cup_{m=1}^{\infty} \mathcal{Z}^m \to \mathcal{Y}^{\mathcal{X}}$ which is parameterized by χ then the model selection problem reduces to finding the the best parameter $\chi(z)$ for a given training sample $z \in \mathcal{Z}^m$.

A typical model selection task which arises in the case of kernel classifiers is the selection of parameters of the kernel function used, for example, choosing the optimal value of σ for RBF kernels (see Table 2.1).

2.5.4 Adaptive Margin Machines

In order to generalize leave-one-out machines we see that the m constraints in equation (2.56) can be rewritten as

$$\begin{aligned} y_i \sum_{\substack{j=1 \\ j \neq i}}^{m} \alpha_j y_j k\left(x_i, x_j\right) + \alpha_i k\left(x_i, x_i\right) &\geq 1 - \xi_i + \alpha_i k\left(x_i, x_i\right) \qquad i = 1, \ldots, m, \\ y_i f\left(x_i\right) &\geq 1 - \xi_i + \alpha_i k\left(x_i, x_i\right) \qquad i = 1, \ldots, m. \end{aligned}$$

Now, it is easy to see that a training point $(x_i, y_i) \in z$ is linearly penalized for failing to obtain a functional margin of $\tilde{\gamma}_i(\mathbf{w}) \geq 1 + \alpha_i k(x_i, x_i)$. In other words, the larger the contribution the training point makes to the decision rule (the larger the value of α_i), the larger its functional margin must be. Thus, the algorithm controls the margin for each training point *adaptively*. From this formulation one

can generalize the algorithm to control regularization through the margin loss. To make the margin at each training point a controlling variable we propose the following learning algorithm:

$$\text{minimize} \quad \sum_{i=1}^{m} \xi_i \tag{2.57}$$

$$\text{subject to} \quad y_i \sum_{j=1}^{m} \alpha_j y_j k\left(x_i, x_j\right) \geq 1 - \xi_i + \lambda \alpha_i k\left(x_i, x_i\right), \quad i = 1, \ldots, m.$$

$$\boldsymbol{\alpha} \geq \mathbf{0}, \boldsymbol{\xi} \geq \mathbf{0}. \tag{2.58}$$

This algorithm—which we call *adaptive margin machines*—can also be viewed in the following way: If an object $x_o \in \boldsymbol{x}$ is an outlier (the kernel values w.r.t. points in its class are small and w.r.t. points in the other class are large), α_o in equation (2.58) must be large in order to classify x_o correctly. Whilst support vector machines use the same functional margin of one for such an outlier, they attempt to classify x_o correctly. In adaptive margin machines the functional margin is automatically increased to $1 + \lambda \alpha_o k\left(x_o, x_o\right)$ for x_o and thus less effort is made to change the decision function because each increase in α_o would lead to an even larger increase in ξ_o and can therefore not be optimal.

Remark 2.39 (Clustering in feature space) *In adaptive margin machines the objects $x_r \in \boldsymbol{x}$, which are representatives of clusters (centers) in feature space $\mathcal{K}$, i.e., those which have large kernel values w.r.t. objects from its class and small kernel values w.r.t. objects from the other class, will have non-zero α_r. In order to see this we consider two objects, $x_r \in \boldsymbol{x}$ and $x_s \in \boldsymbol{x}$, of the same class. Let us assume that x_r with $\xi_r > 0$ is the center of a cluster (w.r.t. the metric in feature space $\mathcal{K}$ induced by the kernel k) and s with $\xi_s > 0$ lies at the boundary of the cluster. Hence we subdivide the set of all objects into*

$$\begin{aligned}
x_i \in C^+ &: \quad \xi_i = 0, y_i = y_r, i \neq r, i \neq s, \\
x_i \in C^- &: \quad \xi_i = 0, y_i \neq y_r, \\
x_i \in I^+ &: \quad \xi_i > 0, y_i = y_r, i \neq r, i \neq s, \\
x_i \in I^- &: \quad \xi_i > 0, y_i \neq y_r.
\end{aligned}$$

We consider the change in $\boldsymbol{\xi}$ if we increase α_r by $\Delta > 0$ (giving $\boldsymbol{\xi}'$) and simultaneously decrease α_s by Δ (giving $\boldsymbol{\xi}''$). From equations (2.57)–(2.58) we know

that

$$
\begin{array}{rrcl@{\qquad}rcl}
x_i \in C^+: & \xi_i' & = & \xi_i\,, & \xi_i'' & \leq & \Delta k\,(x_i, x_s)\,, \\
x_i \in C^-: & \xi_i' & \leq & \Delta k\,(x_i, x_r)\,, & \xi_i'' & = & \xi_i\,, \\
x_i \in I^+: & \xi_i' & \geq & \xi_i - \Delta k\,(x_i, x_r)\,, & \xi_i'' & = & \xi_i + \Delta k\,(x_i, x_s)\,, \\
x_i \in I^-: & \xi_i' & = & \xi_i + \Delta k\,(x_i, x_r)\,, & \xi_i'' & \geq & \xi_i - \Delta k\,(x_i, x_s)\,, \\
x_r: & \xi_r' & \geq & \xi_r - \Delta\,(1-\lambda)\,k\,(x_r, x_r)\,, & \xi_r'' & = & \xi_r + \Delta k\,(x_r, x_s)\,, \\
x_s: & \xi_s' & \geq & \xi_s - \Delta k\,(x_s, x_r)\,, & \xi_s'' & \geq & \xi_s + \Delta\,(1-\lambda)\,k\,(x_s, x_s)\,.
\end{array}
$$

Now we choose the biggest Δ such that all inequalities for $x_i \in \{I^+, I^-, r, r'\}$ become equalities and for $x_i \in \{C^+, C^-\}$ the r.h.s. equals zero. Then, the relative change in the objective function is given by

$$
\frac{1}{\Delta}\sum_{i=1}^{m}\left(\xi_i' + \xi_i'' - \xi_i\right) = \underbrace{\sum_{i\in I^+}\left(k\,(x_i, x_s) - k\,(x_i, x_r)\right)}_{\text{change of intra-class distance}} - \underbrace{\sum_{i\in I^-}\left(k\,(x_i, x_s) - k\,(x_i, x_r)\right)}_{\text{change of inter-class distance}}\,,
$$

where we assume that $k\,(x_r, x_r) = k\,(x_s, x_s)$. Since the cluster centers in feature space $\mathcal{K}$ minimize the intra-class distance whilst maximizing the inter-class distances it becomes apparent that their α_r will be higher. Taking into account that the maximum Δ considerable for this analysis is decreasing as λ increases we see that, for suitable small λ, adaptive margin machines tend to only associate cluster centers in feature space $\mathcal{K}$ with non-zero α's.

2.6 Bibliographical Remarks

Linear functions have been investigated for several hundred years and it is virtually impossible to identity their first appearance in scientific literature. In the field of artificial intelligence, however, the first studies of linear classifiers go back to the early works of Rosenblatt (1958), Rosenblatt (1962) and Minsky and Papert (1969). These w

orks also contains the first account of the perceptron learning algorithm which was originally developed without any notion of kernels. The more general ERM principle underpinning perceptron learning was first formulated in Vapnik and Chervonenkis (1974). In this book we introduce perceptron learning using the notion of version space. This somewhat misleading name comes from Mitchell (1977), Mitchell (1982), Mitchell (1997) and refers to the fact that all classifiers

$h \in V(z)$ are different "versions" of consistent classifiers. Originally, T. Mitchell considered the hypothesis space of logic formulas only.

The method of regularization introduced in Section 2.2 was originally developed in Tikhonov and Arsenin (1977) and introduced into the machine learning framework in Vapnik (1982). The adaptation of ill-posed problems to machine learning can be found in Vapnik (1982) where they are termed *stochastic ill-posed problems*. In a nutshell, the difference to classical ill-posed problems is that the solution $\mathbf{y}$ is a random variable of which we can only observe one specific sample. As a means to solving these stochastic ill-posed problems, Vapnik suggested *structural risk minimization*.

The original paper which proved Mercer's theorem is by Mercer (1909); the version presented in this book can be found in König (1986). Regarding Remark 2.19, the work by Wahba (1990) gives an excellent overview of covariance functions of Gaussian processes and kernel functions (see also Wahba (1999)). The detailed derivation of the feature space for polynomial kernels was first published in Poggio (1975). In the subsection on string kernels we mentioned the possibility of using kernels in the field of Bioinformatics; first approaches can be found in Jaakkola and Haussler (1999b) and Karchin (2000). For a more detailed treatment of machine learning approaches in the field of Bioinformatics see Baldi and Brunak (1998). The notion of string kernels was independently introduced and developed by T. Jaakkola, C. Watkins and D. Haussler in Watkins (2000) and Haussler (1999). A detailed study of support vector machines using these kernels can be found in Joachims (1998) and Lodhi et al. (2001). For more traditional methods in information retrieval see Salton (1968). The Fisher kernel was originally introduced in Jaakkola and Haussler (1999a) and later applied to the problem of detecting remote protein homologizes (Jaakkola et al. 1999). The motivation of Fisher kernels in these works is much different to the one given in this book and relies on the notion of Riemannian manifolds of probability measures.

The consideration of RKHS introduced in Subsection 2.3.3 presents another interesting aspect of kernels, that is, that they can be viewed as regularization operators in function approximation. By noticing that kernels are the Green's functions of the corresponding regularization operator we can directly go from kernels to regularization operators and vice versa (see Smola and Schölkopf (1998), Smola et al. (1998), Smola (1998) and Girosi (1998) for details). The original proof of the representer theorem can be found in Schölkopf et al. (2001). A simpler version of this theorem was already proven in Kimeldorf and Wahba (1970) and Kivinen et al. (1997).

In Section 2.4 we introduced the support vector algorithm as a combination of structural risk minimization techniques with the kernel trick. The first appearance of this algorithm—which has its roots in the early 1960s (Vapnik and Lerner 1963)—is in Boser et al. (1992). The notion of functional and geometrical margins is due to Cristianini and Shawe-Taylor (1999). For recent developments in kernel methods and large margin classifiers the interested reader is referred to Schölkopf et al. (1998) and Smola et al. (2000). The original perceptron convergence theorem (without using kernels) is due to Novikoff (1962) and was independently proved by Block (1962). The extension to general kernels was presented in Aizerman et al. (1964).

In the derivation of the support vector algorithm we used the notion of canonical hyperplanes which is due to Vapnik (1995); for more detailed derivations of the algorithm see also Vapnik (1998), Burges (1998) and Osuna et al. (1997). An extensive study of the computational complexity of the support vector algorithm can be found in Joachims (1999). In the five years an array of different implementations have been presented, e.g., SVM^{light} (Joachims 1998; Osuna et al. 1997), SMO (Platt 1999; Keerthi et al. 1999a; Shevade et al. 1999) and NPA (Keerthi et al. 1999b).

It was noted that without the introduction of soft margins, classifiers found by the support vector algorithm tend to overfit. This was already observed in practice (Cortes 1995; Schölkopf et al. 1995; Osuna et al. 1997; Joachims 1999; Bennett 1998). This tendency is called the *nonrobustness* of the hard margin SVM algorithm—a term which is due to Shawe-Taylor and Cristianini (2000). In order to introduce soft margins we used the hinge loss (due to Gentile and Warmuth (1999)) whose relation to support vector machines was shown in Sollich (2000). The seminal paper, which introduced the linear soft margin algorithm is Cortes and Vapnik (1995); it also mentions the possibility of quadratically penalizing the slacks. The empirical success of quadratic soft margin support vector machines has been demonstrated in Veropoulos et al. (1999) and Brown et al. (2000). The former paper also noted that different values of λ for training points from different classes can be used to compensate for unequal class probabilities (see also Osuna et al. (1997) for details). Experimental evidence of the advantage of normalizing training data in feature space before applying the support vector algorithm can be found in Schölkopf et al. (1995), Joachims (1998) and Joachims (1999); theoretical evidence is given in Herbrich and Graepel (2001b).

It is interesting to remark that the research on linear classifiers has run rather parallel in the computer science and the statistical physics community (see Guyon and Storck (2000) for a recent overview). One of the earliest works about support

vector machines (which are called *maximal stability perceptrons*) is by Lambert (1969). After this work, many statistical physicists got involved in neural networks (Gardner 1988; Gardner and Derrida 1988). As a consequence, several large margin alternative of the perceptron learning algorithm were devised, for example, the *minimal overlap* (MinOver) algorithm (Krauth and Mézard 1987) or the *adatron* (Anlauf and Biehl 1989). Finally, a fast primal-dual method for solving the maximum margin problem has been published in Ruján (1993).

In Subsection 2.4.4 several extensions of the original support vector algorithm are presented. For more details on the extension to multiple classes see Weston and Watkins (1998), Platt et al. (2000), Hastie and Tibshirani (1998), Guermeur et al. (2000) and Allwein et al. (2000). There exits a vast literature on support vector regression estimation; for an excellent overview see Smola and Schölkopf (2001), Smola (1996), Smola (1998) and Smola and Schölkopf (1998). It has also been shown that support vector machines can be applied to the problem of density estimation (Weston et al. 1999; Vapnik and Mukherjee 2000). The reparameterization of the support vector algorithm in terms of ν, the fraction of margin errors, was first published in Schölkopf et al. (2000) where it was also applied to the support vector algorithm for regression estimation.

Finally, in Section 2.5, we introduce the leave-one-out error of algorithms which motivate an algorithm called adaptive margin machines (Weston and Herbrich 2000). The proof of the unbiasedness of the leave-one-out error can be found in Lunts and Brailovsky (1969) and also in Vapnik (1998, p. 417). The bound on the leave-one-out error for kernel classifiers presented in Theorem 2.37 was proven in Jaakkola and Haussler (1999b).