Parzen windows estimators of the two class densities,

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

Parzen Windows

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

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

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

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

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

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

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

1.3 Some Insights From Statistical Learning Theory

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

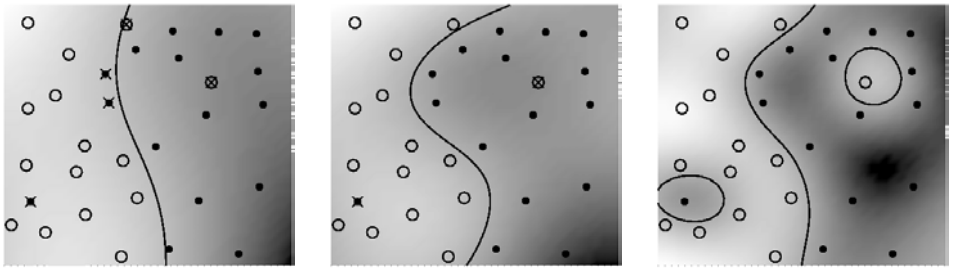

Figure 1.2 2D toy example of binary classification, solved using three models (the decision boundaries are shown). The models vary in complexity, ranging from a simple one *(left)*, which misclassifies a large number of points, to a complex one *(right)*, which "trusts" each point and comes up with solution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

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

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

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

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

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

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

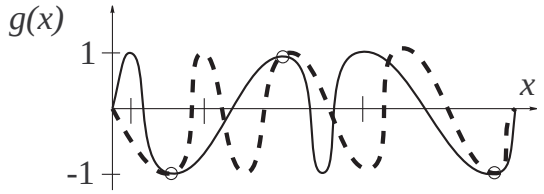

Figure 1.3 A 1D classification problem, with a training set of three points (marked by circles), and three test inputs (marked on the x-axis). Classification is performed by thresholding real-valued functions $g(x)$ according to $\mathrm{sgn}(f(x))$. Note that *both* functions (dotted line, and solid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

that the learning machine can implement.

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

IID Data

Loss Function

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

Test Data

Empirical Risk

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

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

Risk

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

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

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

Capacity

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

VC dimension

Shattering

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

VC Bound

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

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

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

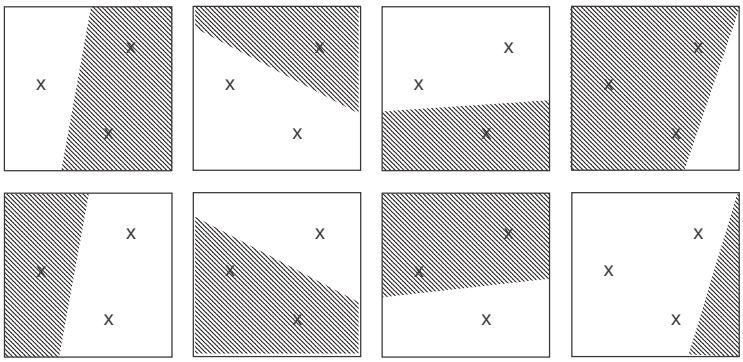

Figure 1.4 A simple VC dimension example. There are $2^3 = 8$ ways of assigning 3 points to two classes. For the displayed points in $\mathbb{R}^2$, all 8 possibilities can be realized using separating hyperplanes, in other words, the function class can shatter 3 points. This would not work if we were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

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

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

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

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

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

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

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

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

1.4 Hyperplane Classifiers

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

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

corresponding to decision functions

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

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

Optimal Hyperplane

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

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

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