Chapter 2

Background

Bayes decision theory provides a fundamental statistical approach to the problem of pattern classification. It is based on the assumption that the decision problem is posed in probabilistic terms, and that all relevant probability values are known. In the following we introduce *supervised learning* approaches to estimating the posterior probability values, in an attempt to come as closer as possible to the optimum Bayesian classifier. The chapter concludes with a discussion on the main concepts and properties of support vector machine classifiers.

2.1 Classification

In a classification problem an observation is characterized by q feature measurements $\mathbf{x} = (x_1, \cdots, x_q) \in \Re^q$ and is presumed to be a member of one of J classes, $L_j, j = 1, \ldots, J$. The particular group is unknown, and the goal is to assign the given object to the correct group using its measured values $\mathbf{x}$.

Let $\lambda(j|k)$ be the loss incurred by assigning $\mathbf{x}$ to the jth group L_j, when it is actually a member of the kth group L_k [DH73]. By denoting $P(k|\mathbf{x})$ the probability that $\mathbf{x}$ is a member of the kth class given the particular set of measurements $\mathbf{x}$, the *expected loss* associated with class j is

$$R(j|\mathbf{x}) = \sum_{k=1}^{J} \lambda(j|k) P(k|\mathbf{x}), \tag{2.1}$$

which is also known as the *conditional risk*. Whenever we encounter a particular observation $\mathbf{x}$, we can minimize the expected loss by selecting the class that minimizes the conditional risk,

$$j^* = \arg \min_{1 \le j \le J} R(j|\mathbf{x}). \tag{2.2}$$

A loss function of particular interest is the so-called *symmetrical* or *zero-one* loss function,

$$\lambda(j|k) = \begin{cases} 0 & \text{if } j = k \\ 1 & \text{if } j \neq k \end{cases} \tag{2.3}$$

where $j, k = 1, \cdots, J$. This loss function assigns no loss to a correct decision and a unit loss to any error. Thus, all errors are equally costly. The risk corresponding to this loss function is the average probability of error, or the *error rate*, since the conditional risk is

$$\begin{aligned} R(j|\mathbf{x}) &= \sum_{k=1}^{J} \lambda(j|k) P(k|\mathbf{x}) \\ &= \sum_{k \neq j} P(k|\mathbf{x}) \\ &= 1 - P(j|\mathbf{x}), \end{aligned} \tag{2.4}$$

and $P(j|\mathbf{x})$ is the conditional probability that class j is correct. Therefore, equation (2.2) reduces to

$$j^* = \arg \max_{1 \le j \le J} P(j|\mathbf{x}) \tag{2.5}$$

if all misclassifications are considered equally costly. Equations (2.2) and (2.5) are known as the Bayes decision rules and their associated error rate is the minimum achievable. Thus, to minimize the average probability of error, we should select the class j that *maximizes* the posterior probability $P(j|\mathbf{x})$. In other words, for *minimum error rate*:

$$Decide \quad j \;\; if \;\; P(j|\mathbf{x}) > P(k|\mathbf{x}) \;\; for\; all\; k \neq j.$$

In order to apply the Bayes decision rule the true conditional probabilities $\{P(j|\mathbf{x})\}_{j=1}^{J}$ must be known for every query point $\mathbf{x} \in \Re^q$ at which class predictions are to be made. Unfortunately, this is seldom the case, and the conditional probabilities must be estimated.

Bayes decision theory provides a fundamental statistical approach to the problem of pattern classification. It is based on the assumption that the decision problem is posed in probabilistic terms, and that all relevant probability values are known. In the following we introduce *supervised learning* approaches to estimating the posterior probability values, in an attempt to come as closer as possible to the optimum Bayesian classifier.

We are given J classes and N training observations. The training observations consist of q feature measurements $\mathbf{x} = (x_1, \cdots, x_q) \in \Re^q$ and the known class labels, $j = 1, \ldots, J$:

$$\{(\mathbf{x}_n, y_n)\}_{n=1}^{N} \tag{2.6}$$

Here $y_n \in \{1, \ldots, J\}$. The goal is again to predict the class label of a given query $\mathbf{x}_0$. The training data (2.6) are used to obtain the estimates

$$\{\hat{P}(j|\mathbf{x}_0)\}_{j=1}^{J} \tag{2.7}$$

which are then used in (2.1) and (2.2) to get the class assignment estimate

$$\hat{j}(\mathbf{x}_0) = \arg \min_{1 \leq j \leq J} \sum_{k=1}^{J} \lambda(j|k) \hat{P}(k|\mathbf{x}_0), \tag{2.8}$$

or the corresponding analog for (2.5):

$$\hat{j}(\mathbf{x}_0) = \arg \max_{1 \leq j \leq J} \hat{P}(j|\mathbf{x}_0). \tag{2.9}$$

2.2 Density Estimation

One approach to estimating the posterior class conditional probabilities is based on density estimation via Bayes theorem

$$P(j|\mathbf{x}) = \frac{p(\mathbf{x}|j)P(j)}{\sum_{k=1}^{J} p(\mathbf{x}|k)P(k)}. \tag{2.10}$$

Here $p(\mathbf{x}|j)$ is the probability density function of $\mathbf{x}$ given class j, and $P(j)$ is the probability that reflects our *prior* knowledge of observing an object of class j in the absence of a set of measurement values $\mathbf{x}$. Thus, Bayes rule shows how observing the values $\mathbf{x}$ changes the a priori probability $P(j)$ to the posterior probability $P(j|\mathbf{x})$.

With this approach, the training data in each class j, for $j = 1, \cdots, J$, are used separately to estimate the corresponding probability density function $p(\mathbf{x}|j)$ over the measurement space. The prior probabilities $P(j)$ are either known in advance through some knowledge of the nature of the problem under study, or are estimated as the proportions of each class in the training data set. These estimates are then used to derive the posterior class conditional probabilities (2.7) through (2.10).

2.2.1 Discriminant Analysis

An example of the density estimation approach is *discriminant analysis* [Mcl92]. In this case, the class conditional density functions are approximated by Gaussian distributions, and the training data for each class are used to estimate the mean vector and the covariance matrix for that class. The goal of discriminant analysis is to find *discriminant functions* in feature space that achieve the lowest error rate to the extent that the normal assumption is correct. Thus, a classifier is defined in terms of a set of discriminant functions $g_j(\mathbf{x})$, $j = 1, \cdots, J$. The classifier assigns a feature vector $\mathbf{x}$ to class j if

$$g_j(\mathbf{x}) > g_k(\mathbf{x}) \quad \forall\, k \neq j.$$

In words, the classifier selects the class corresponding to the largest discriminant. A Bayes classifier can easily be represented in this way. We can define $g_j(\mathbf{x}) = -R(j|\mathbf{x})$, since the maximum discriminant function will then correspond to the minimum conditional risk. For the minimum error rate case, things could be further simplified by taking $g_j(\mathbf{x}) = P(j|\mathbf{x})$, so that the maximum discriminant function corresponds to the maximum posterior probability. The discriminant functions can be written in a variety of forms, all resulting in the same classification result. In general, by replacing every $g_j(\mathbf{x})$ with $f(g_j(\mathbf{x}))$, where f is a monotonically increasing function, resulting classification is unchanged. In particular, for minimum error rate classification, it is convenient to consider

$$g_j(\mathbf{x}) = \log p(\mathbf{x}|j) + \log P(j) \tag{2.11}$$

as a discriminant function, which is equivalent to having $g_j(\mathbf{x}) = P(j|\mathbf{x})$. The expression in (2.11) can be evaluated if we assume that the densities $p(\mathbf{x}|j)$ are multivariate normal. Let $p(\mathbf{x}|j) \sim$

$N(\boldsymbol{\mu}_j, \Sigma_j)$, where

$$N(\boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{q/2}|\Sigma|^{1/2}} exp[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^t \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})]. \quad (2.12)$$

Then,

$$g_j(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^t \Sigma_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j) - \frac{q}{2}\log 2\pi - \frac{1}{2}\log|\Sigma_j| + \log P(j). \quad (2.13)$$

A simple case arises when the covariance matrices for all classes are identical. Since $|\Sigma_j|$ in (2.13) becomes independent of j, it can be ignored, along with the constant $(q/2)\log 2\pi$, resulting in the discriminat functions

$$g_j(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^t \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_j) + \log P(j). \quad (2.14)$$

If the a priori probabilities $P(j)$ are the same for all J classes, then the term $\log P(j)$ can be ignored. In this case, the decision rule can be stated very simply: To classify a feature vector $\mathbf{x}$, measure the distance $(\mathbf{x}-\boldsymbol{\mu}_j)^t\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)$ (known as the *squared Mahalanobis distance*) from $\mathbf{x}$ to each of the J mean vectors, and assign $\mathbf{x}$ to the category of the nearest mean. Unequal a priori probabilities bias the decision towards the a priori more likely class.

By expanding the term $(\mathbf{x}-\boldsymbol{\mu}_j)^t\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)$ it can be shown that the quadratic term $\mathbf{x}^t\Sigma^{-1}\mathbf{x}$ is independent of j. By deleting it, we obtain linear discriminant functions:

$$g_j(\mathbf{x}) = \mathbf{w}_j^t\mathbf{x} + w_{j0}, \quad (2.15)$$

where

$$\mathbf{w}_j = \Sigma^{-1}\boldsymbol{\mu}_j \quad (2.16)$$

and

$$w_{j0} = -\frac{1}{2}\boldsymbol{\mu}_j^t\Sigma^{-1}\boldsymbol{\mu}_j + \log P(j). \quad (2.17)$$

Thus, the resulting decision boundaries are hyperplanes. They define the optimal decision rule when the normal and equal covariance assumptions are satisfied by the actual underlying distributions of the data.

2.2.2 Naive Bayesian Classifier

The so-called “naive” Bayesian classifier makes the assumption that the values of the attributes of an example are independent given the class of the example. As a consequence, the joint density function in (2.10) can be written as the product of the marginal densities of each attribute, making the use of Bayes theorem much simpler. Although this assumption is almost always violated in practice, naive Bayesian learning is remarkably effective in practice [DP96].

2.3 Regression

The density estimation approach to classification computes posterior probabilities by estimating the probability density functions conditioned on the value of the class label. A second category of supervised learning techniques directly estimate the probabilities $\{P(j|\mathbf{x})\}_{j=1}^{J}$. At a given query point $\mathbf{x}$ the class label y is assumed to be a random variable from a multinomial distribution with probabilities $\{P(j|\mathbf{x})\}_{j=1}^{J}$. Each possible value for y at $\mathbf{x}$ is characterized by an additional output variable g_j such that

$$g_j|\mathbf{x} = \begin{cases} 1 & \text{if } y = j \\ 0 & \text{otherwise} \end{cases}$$

where $j = 1, \cdots, J$. $g_j|\mathbf{x}$ represents the characteristic function of class j at $\mathbf{x}$ and gives rise to J separate training sets, one for each class j:

$$\{\mathbf{x}_n, g_j\}_{n=1}^N, \quad j = 1, \cdots, J. \tag{2.18}$$

Then

$$f_j(\mathbf{x}) \overset{\text{def}}{=} P(j|\mathbf{x}) = P(g_j = 1|\mathbf{x}) = E[g_j|\mathbf{x}]. \tag{2.19}$$

The posterior probabilities are the target functions $f_j(\mathbf{x})$ to be estimated by using the corresponding training samples (2.18)

$$f_j(\mathbf{x}) = \arg \min_f E[(g_j - f)^2|\mathbf{x}], \tag{2.20}$$

where $j = 1, \cdots, J$, so that they become the solutions of a set of least-squares problems. It is evident that, since they represent probabilities, the target functions $\{f_j(\mathbf{x})\}_{j=1}^J$ satisfy the constraints

$$0 \leq f_j(\mathbf{x}) \leq 1, \quad and \quad \sum_{j=1}^J f_j(\mathbf{x}) = 1.$$

Many of the techniques developed in machine learning and pattern recognition apply more or less closely this regression paradigm to the classification problem. We elaborate more on some specific methods such as *neural networks, decision trees, K-nearest-neighbors,* and *support vector machines*.

2.3.1 Neural Networks: Generalized Linear Models (GLIM)

We describe how a two class classification problem can be modeled by using a neural network representation [Bis95]. The interpretation of the output of the resulting neural network model as posterior probabilities has a statistical foundation in Bayes theory [Jor95].

Suppose the two classes have labels 0 and 1. For a given point $\mathbf{x}$, its class label y is assumed to be a random variable from a Bernoulli distribution with probabilities $\{P(j|\mathbf{x})\}_{j=0}^{1}$:

$$p(y;\mu) = \mu^y(1-\mu)^{1-y} \tag{2.21}$$

where $\mu = P(1|\mathbf{x})$ is the posterior probability associated with class 1, $P(0|\mathbf{x}) = (1-\mu)$, and $y \in \{0,1\}$. Equation (2.21) represents the Bernoulli mass function.

Our goal is to estimate the posterior probability value μ. We are given a training set $X = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. Here $y_i = \{0,1\}$. The learning system needs to infer the probability value μ by the given pairs $(\mathbf{x}_i, y_i)$ in X. We can write $\mu = P(1|\mathbf{x}) = f(\mathbf{w}^t\mathbf{x})$, which means: given $\mathbf{x}$ as input, the corresponding probability of observing the output $y = 1$ is $P(1|\mathbf{x}) = f(\mathbf{w}^t\mathbf{x})$. Here $\mathbf{x}$ undergoes a linear transformation, whose result is then mapped to the interval $[0,1]$ by a non linear "squashing" function f. The parameters (*weights*) of the linear transformation $\mathbf{w} = (w_1, \cdots, w_q)$ are not known and are to be estimated by the learning system.

By assuming independence and identical distribution among the data in X, we can write the joint density function of X as the product of the densities of each data pair:

$$L(\mathbf{w}, X) = \prod_{i=1}^{N} \mu_i^{y_i}(1-\mu_i)^{1-y_i}, \tag{2.22}$$

where $\mu_i = f(\mathbf{w}^t\mathbf{x}_i)$. Equation (2.22) represents the *likelihood* function of X. Our objective is to maximize $L(\mathbf{w}, X)$ with respect to $\mathbf{w}$. By computing the log of (2.22) we obtain $\log L(\mathbf{w}, X) = l(\mathbf{w}, X) = \sum_{i=1}^{N}[y_i \log \mu_i + (1-y_i)\log(1-\mu_i)]$, and thus the cost function

$$J(\mathbf{w}) = -(\sum_{i=1}^{N}[y_i \log \mu_i + (1-y_i)\log(1-\mu_i)]), \tag{2.23}$$

which is the *binary cross entropy*. By taking the gradient with respect to the weights w_j we get

$$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{N} (y_i - \mu_i) x_{ij}, \tag{2.24}$$

where x_{ij} is the jth component of $\mathbf{x}_i$, and $f(z)$ is the *logistic function*:

$$f(z) \overset{\text{def}}{=} \frac{1}{1 + e^{-z}}. \tag{2.25}$$

Finally, by considering the stocastic gradient, we obtain the following learning rule for the weights

$$\Delta w_j \propto (y - \mu) x_j. \tag{2.26}$$

The use of the logistic function as the output function for a two case classification problem has a motivation within Bayes theory: given specific assumptions, the logistic function formally provides the posterior probability of a given class [Jor95]. A similar result can be obtained for the general classification problem with more than two groups, where the logistic function is replaced by the *softmax function* and the cost function gets the general form of a cross entropy.

2.3.2 Decision Trees

Decision trees constitute a local learning paradigm that employs local averaging to estimate the class posterior probabilities for the decision rule (2.8) [BFO+84, Qui86, Qui93]. The regions over which the averaging takes place are constructed in a highly adaptive manner by using a top-down recursive splitting strategy, from which the alternative name of *recursive partitioning* (RP) is derived. RP begins with a single region R_0 containing all the training data. At each step every existing region is split into two subregions, thereby increasing the number of regions. This recursive splitting procedure is continued until a region meets a local terminal criterion, i.e. it contains training observations of the same class, and is not further split. When all regions have met the terminal criterion, they provide the final input space partition for local estimation of class probabilities.

The shape of the terminal regions is governed by the splitting procedure. One defines a splitting function $g(\mathbf{x}, \mathbf{a})$ of the input variables $\mathbf{x}$, characterized by a set of parameters $\mathbf{a}$, and a real valued split point s. The form of the split function is usually taken to be linear

$$g(\mathbf{x}, \mathbf{a}) = \mathbf{a}^t \mathbf{x} \tag{2.27}$$

(CART) [BFO+84], often with the restriction of being parallel to the coordinate input axes (CART, C4.5) [BFO+84, Qui93]

$$\mathbf{a} \in \{\mathbf{e}_1, \cdots, \mathbf{e}_n\}. \tag{2.28}$$

Here $\mathbf{e}_j$ is the basis unit vector for the jth input coordinate. The particular variable j and location s used to define the split are those that jointly minimize a given criterion. In general, such criterion is related to the average misclassification risk associated with using the resulting subregions to classify all training observations that respectively lie within them. In other words, the variable j and location s that best separate the data in the current region are chosen.

After the splitting is completed a “pruning” procedure is usually applied that recursively recombines adjacent regions in a bottom-up manner using cross-validated misclassification risk to determine when to stop the pruning. Other approaches are also possible. For example, all available data can be used for training, and a statistical test (e.g., chi-square) is then applied to estimate whether pruning a particular node is likely to produce an improvement beyond the training set. Techniques that stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data, are also in use [Mit97].

The resulting set of regions represents a disjoint partition of the input measurement space. The region used for estimating the class probabilities and making the assignment (2.8) for any prediction point $\mathbf{x}_0$ is the resulting region in which the point lies.

We note that at each step in the recursive subdivision the variable that gives the most estimated classification information is considered for splitting, thereby reducing the extent of the neighborhood along the chosen axis for all successive regions that are descendants of the one being split. Moreover, the choice of the variable for splitting is based only on the local data contained in each region. Thus, recursive partitioning methods can exploit "local relevance" of input variables at different query points $\mathbf{x}_0$ in feature space. Some recursive partitioning implementations add locally derived variables that are functions of the original input variables, allowing regions to have non-axis oriented boundaries. Finally, if pruning is employed recursive partitioning methods also estimates the best region size locally for different query points $\mathbf{x}_0$.

With this built-in flexibility one might expect recursive partitioning methods to perform well in general. However, in benchmark studies even the simplest K-NN rules (see Section 2.3.3) often outperform recursive partitioning methods. Among the possible reasons for this behavior, some can be identified. The training sample size N restricts the number of splits that can be used to limit the extension of a subregion. As the partitioning proceeds the number of training observations in successive regions becomes smaller and smaller. After a relatively small number of splits the data remaining is insufficient to provide meaningful estimates. A second limitation, related to the first, is that decision trees produce a *partition* of the input space, that is the terminal regions that cover the space are disjoint. The consequence is that the resulting approximations are piecewise-constant and discontinuous, leading to large errors (bias) near region boundaries. Each prediction point is contained in only one region and it can be very far from the region center and training points in that region. This can induce high bias that may overcome the bias reduction achieved by customizing the shape of the region. Finally, the approximations produced by RP are highly unstable with respect to minor perturbations of the training data. This leads to high variance predictions, induced by sampling fluctuations of the mechanism that produces the training data. Minor changes in an early split can have a major impact on later splits producing very different terminal regions.

Nearest neighbor methods, to be introduced next, overcome some of the limitations of recursive partitioning techniques by producing continous and overlapping neighborhoods, resulting in highly stable procedures with respect to perturbations of the training data. It should be also mentioned that several approaches have been proposed to mitigate the instability of RP methods. Among them is the *bagging* procedure [Bre96, Qui96].

2.3.3 Nearest-Neighbor Methods

The K nearest neighbor classification method [CD88, Ho98, Low95, Mcl92, Sal91, Sto77] is a simple and appealing approach: it finds the K nearest neighbors of the query point $\mathbf{x}_0$ in the training set, and then predicts the class label of $\mathbf{x}_0$ as the most frequent one occurring in the K neighbors. Such a method produces continuous and overlapping neighborhoods, and uses a different neighborhood for each individual query so that all points in the neighborhood are close to the query, to the extent possible. It is based on the assumption of smoothness of the target functions, which translates to locally constant class posterior probabilities for a classification problem. That is, $f_j(\mathbf{x} + \delta\mathbf{x}) \simeq f_j(\mathbf{x})$ for $||\delta\mathbf{x}||$ small enough, where $\{f_j(\mathbf{x})\}_{j=1}^J = \{P(j|\mathbf{x})\}_{j=1}^J$. Then,

$$f_j(\mathbf{x}_0) \simeq \frac{1}{|N(\mathbf{x}_0)|} \sum_{\mathbf{x} \in N(\mathbf{x}_0)} f_j(\mathbf{x}) \tag{2.29}$$

where $N(\mathbf{x}_0)$ is a neighborhood of $\mathbf{x}_0$ that contains points $\mathbf{x}$ in the q dimensional space that are "close" to $\mathbf{x}_0$. $|N(\mathbf{x}_0)|$ denotes the number of points in $N(\mathbf{x}_0)$. Given the training data $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$, this motivates the estimates

$$\hat{f}(j|\mathbf{x}_0) = \frac{\sum_{n=1}^N 1(\mathbf{x}_n \in N(\mathbf{x}_0))1(y_n = j)}{\sum_{n=1}^N 1(\mathbf{x}_n \in N(\mathbf{x}_0))}, \tag{2.30}$$

where $1(\cdot)$ is an indicator function such that it returns 1 when its argument is true, and 0 otherwise.

A particular nearest-neighbor method is defined by how the neighborhood $N(\mathbf{x}_0)$ is specified. K-nearest-neighbor methods (K-NN) define the region at $\mathbf{x}_0$ to be the one that contains exactly the K closest training points to $\mathbf{x}_0$ according to a p-norm distance metric on the Euclidean space of the input measurement variables

$$D_p(\mathbf{x}_0, \mathbf{x}) = \{\sum_{i=1}^q |[W(\mathbf{x}_0)(\mathbf{x}_0 - \mathbf{x})]_i|^p\}^{1/p}. \tag{2.31}$$

The resulting neighborhood is determined by the value of K and by the choice of the distance measure, which in turn depends on a norm $p > 0$ and a metric defined by the matrix $W(\mathbf{x}_0) \in \Re^{q \times q}$.

The K-nearest-neighbor method has nice asymptotic properties [CH67, Cov68]. As the training sample size N becomes arbitrarily large it results

$$\lim_{N \to \infty} \hat{f}_j(\mathbf{x}) = f_j(\mathbf{x}) \tag{2.32}$$

provided that the value for K is chosen as a function of N so that

$$\lim_{N \to \infty} K = \infty, \qquad \lim_{N \to \infty} K/N = 0. \tag{2.33}$$

The first condition reduces the variance by making the estimation independent of the accidental characteristics of the K nearest neighbors. The second condition reduces the bias by assuring that the K nearest neighbors are arbitrarily close to the query point. Intuitively, if K is kept fixed when the number N of samples is allowed to approach infinity, then all the K nearest neighbors will converge to $\mathbf{x}$. Conditions (2.33) require that K grow infinitely large, but at a lower rate than N.

These results imply that asymptotically, as $N \to \infty$, the K-NN rule achieves the minimum error rate of the Bayes rule, provided that the value of K is properly chosen, independent of the norm p or the metric $W(\mathbf{x})$.

Another asymptotic result holds for the 1-NN rule. Let E_1 be the classification error rate of the 1-NN rule and E_∞ that of the Bayes rule. Then $\lim_{N\to\infty} E_1 = e$, where $E_\infty \leq e \leq E_\infty(2 - e) \leq 2E_\infty$, so that with an unlimited number of samples the error rate for the 1-NN rule is no worse than twice the Bayes rate. This implies that in the asymptotic limit no decision rule is more than twice as accurate as the 1-NN rule, including a K-NN rule for any value of K.

Let us gain some understanding on why the 1-NN rule should work well. Let $\mathbf{x}_0$ be the query point and $\mathbf{x}$ its nearest neighbor in the training data. We recall that the label y of $\mathbf{x}$ is a random variable, and the probability that $y = j$ is simply the posterior probability $P(j|\mathbf{x})$. Then, when the number of samples is very large, it is reasonable to assume that $\mathbf{x}$ is sufficiently close to $\mathbf{x}_0$ so that $P(j|\mathbf{x}) \simeq P(j|\mathbf{x}_0)$. In this prospective we can view the 1-NN rule as a randomized decision that classifies $\mathbf{x}_0$ by selecting the class j with probability $P(j|\mathbf{x}_0)$.

If we define $j^*(\mathbf{x})$ as

$$P(j^*|\mathbf{x}) = \max_j P(j|\mathbf{x})$$

then the Bayes rule always selects j^*. When $P(j^*|\mathbf{x})$ is close to one, the nearest-neighbor selection is almost always the same as the Bayes selection. That is, when the minimum probability of error is small, the nearest-neighbor probability of error is also small. When $P(j^*|\mathbf{x})$ is close to $1/J$, so that all classes are essentially equally likely, the selections made by the nearest-neighor rule and the Bayes rule are rarely the same, but the probability of error is approximately $1 - 1/J$ for both.

Example. Consider two classes (labelled 1 and 2) with probability density functions f_1 and f_2, respectively, as in Figure 2.1. Each density function is a mixture of two uniform distributions. Both densities share the same component for $x \in [3, 5]$; in this interval both densities have value $p < 1/2$. Assume that the prior probabilities of the two classes are equal to 1/2. The Bayes risk at x, when $x \in [3, 5]$, is

$$r^*(x) = \sum_{j=1}^{2} P(j|x)(1 - P(j|x)) = 1/2$$

Since $x \in [3, 5]$ with probability $2p$, the Bayes risk is equal to p. If $x \in [1, 2]$, it definetely belongs to class 1. With probability 1, the training set contains at least one sample of class 1 in the interval $[1, 2]$. This is because the training set size is growing to infinity. Hence, the 1-NN rule will classify x correctly. A similar argument applies when $x \in [6, 7]$. However, when $x \in [3, 5]$, $P(1|x) = 1/2$, that is the conditional probability that the class label is 1 given x is 1/2, because the priors are equals, and the class conditional densities are equal in such interval. Therefore, no matter how the 1-NN classifies a sample $x \in [3, 5]$, it is wrong with probability 1/2, just like the Bayes decision rule. Hence, the 1-NN rule has the same conditional probability of error as the Bayes decision rule for every x, and therefore its risk equals the Bayes risk.

The asymptotic results discussed suggest that the 1-NN based on simple Euclidean distance ($p = 2$, $W(\mathbf{x}_0) = I$)

$$D_2(\mathbf{x}_0, \mathbf{x}) = [\sum_{i=1}^{q} (x_{0i} - x_i)^2]^{1/2} \tag{2.34}$$

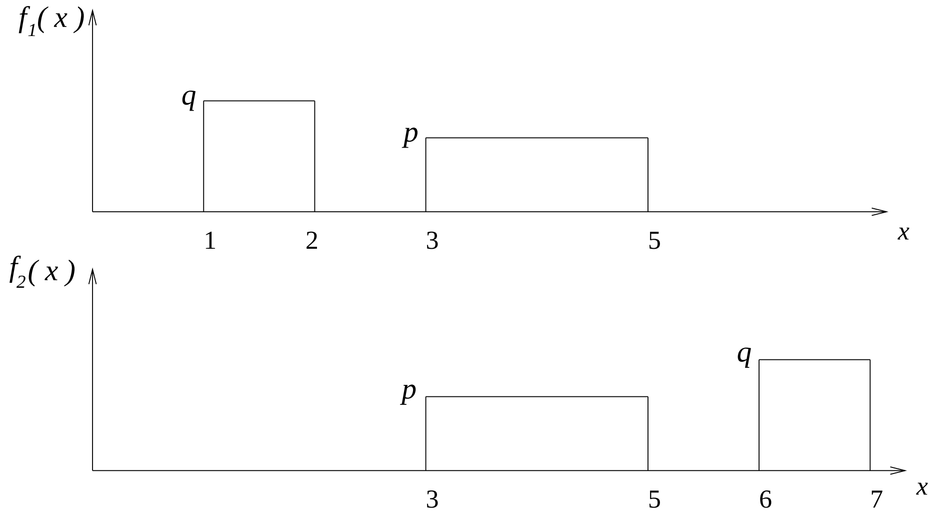

Figure 2.1: Probability density functions $f_1(x)$ and $f_2(x)$.

might perform well provided that the training data set is not too small.

An obvious question concerns the rate of convergence of the 1-NN procedure: how rapidly the performance of the 1-NN rule converges to the asymptotic value? Unfortunately, the only statements that can be made in the general case are negative. It can be shown that convergence can be arbitrarily slow, and the error rate need not even decrease monotonically with the increasing number of data [Cov68]. As with other nonparametric methods, it is difficult to obtain anything other than asymptotic results without making regularity assumptions about the underlying probability structure. Under suitable smoothness conditions, the error rate of the N sample nearest neighbor rule converges to its limit on the order of $1/N^2$: $E_1(N) = E_\infty + O(1/N^2)$ [Cov68].

2.4 Support Vector Machines

In this section we introduce the main concepts and properties of *support vector machines* (SVMs) [Bur98, Vap98, Vap99, CT01, SS02]. Again, we are given N observations. Each observation consists of a pair: a vector $\mathbf{x}_i \in \Re^q$, $i = 1, \ldots, N$, and the associated class label $y_i \in \{-1, 1\}$ (we restrict ourselves to a two class case). It is assumed that there exists some unknown probability distribution $P(\mathbf{x}, y)$ from which these data are drawn. The task it to learn the set of parameters α in $f(\mathbf{x}, \alpha)$ so that f realizes the mapping $\mathbf{x}_i \to y_i$. A particular choice of α defines the corresponding trained machine $f(\mathbf{x}, \alpha)$.

The expectation of the test error (i.e., the expected risk, or just the risk) for a trained machine is

$$R(\alpha) = \int \frac{1}{2}|y - f(\mathbf{x}, \alpha)| dP(\mathbf{x}, y).$$

This quantity gives a nice way of writing the true mean error, but unless we have an estimate of what $P(\mathbf{x}, y)$ is, it is not very useful. The empirical risk $R_{emp}(\alpha)$ is then defined, as the mean error rate measured over the training set:

$$R_{emp}(\alpha) = \frac{1}{2N} \sum_{i=1}^{N} |y_i - f(\mathbf{x}_i, \alpha)|.$$

The following bound holds (with high probability over the random draw of the training sample) [Vap99]:

$$R(\alpha) \leq R_{emp}(\alpha) + Conf(h),$$

where h is the Vapnik Chervonenkis (VC) dimension, and is a measure of the ability of the machine to learn any training set without error. The term $Conf(h)$ is called the VC confidence. Given a family of functions $f(\mathbf{x}, \alpha)$, it is desirable to choose the machine which gives the lowest upper bound on the risk. The first term, $R_{emp}(\alpha)$, represents the accuracy attained on a particular training set, whereas the second term $Conf(h)$ represents the ability of the machine to learn any training set without error. $R_{emp}(\alpha)$ and $Conf(h)$ respectively drive the bias and the variance of the generalization error. The best generalization error is achieved when the right balance between these two terms is attained. This gives a principled method for choosing a learning machine for a specific task, and is the essential idea of structural risk minimization.

Unlike traditional methods which minimize the empirical risk, a support vector machine aims at minimizing the above upper bound of the generalization error. It achieves this goal by learning the αs in $f(\mathbf{x}, \alpha)$ so that the resulting trained machine satifies the maximum margin property, i.e. the decision boundary it represents has the maximum minimum distance from the closest training point.

The well developed theory that has motivated SVMs makes them an attractive learning machine for a variety of tasks. An SVM maps the data into a higher-dimensional space (feature space) and defines a separating hyperplane there. Translating the training set into a higher-dimensional space incurs both computational and learning-theoretic costs. SVMs avoid overfitting by choosing a particular hyperplane among the many that can separate the data in the feature space, specifically the maximum margin hyperplane.

The computational burden of explicitly representing the feature vectors is avoided by defining a function, called the *kernel function*, that plays the role of the dot product in feature space. Therefore, an SVM can locate a separating hyperplane in feature space and classify points in that space without ever representing the space explicitly.

By choosing different functions as kernels, SVMs can realize Radial Basis Function (RBF), Polynomial and Multi-layer Perceptron classifiers. Compared with the traditional way of implementing such classifiers, SVMs have the advantage of automatically selecting both the number and locations of the kernel function during training.

In addition to avoid overfitting, the use of the maximum margin hyperplane leads to a learning algorithm that can be reduced to a convex optimization problem. In order to train the SVM, the unique minimum of a convex function must be found. As a consequence, support vector machines do not suffer from the local minima problem that affects many learning schemes and, unlike the backpropagation learning algorithm for neural networks, a given SVM will always deterministically converge to the same solution for a given data set, regardless of the initial conditions.

Another appealing feature of SVMs is the sparseness representation of the decision boundary they provide. The location of the separating hyperplane in feature space is specified via real-valued weights on the training examples. In general, those training examples that lie far away from the hyperplane do not participate in its specification and therefore receive zero weight. Training examples that lie close to the decision boundary between the two classes receive non-zero weights. These training examples are called *support vectors*, since their removal would change the location of the separating hyperplane. The design of SVMs, in general, allows the number of support vectors to be small compared to the total number of training examples. This property allows the SVM to classify new examples efficiently, since the majority of the training examples will be safely ignored.

2.4.1 Learning with SVMs

In the simple case of two linearly separable classes, a support vector machine selects, among the infinite number of linear classifiers that separate the data, the classifier that minimizes an upper bound on the generalization error. The SVM achieves this goal by computing the classifier that satifies the maximum margin property, i.e. the classifier whose decision boundary has the maximum minimum distance from the closest training point.

If the two classes are non-separable, the SVM looks for the hyperplane that maximizes the margin and that, at the same time, minimizes a quantity proportional to the number of misclassification errors. The trade-off between margin and misclassification error is driven by a positive constant C that has to be chosen beforehand. The corresponding decision function is then obtained by considering the $sign(f(\mathbf{x}))$, where $f(\mathbf{x}) = \sum_i \alpha_i y_i \mathbf{x}_i^T \cdot \mathbf{x} - b$, and the coefficients α_i are the solution of a convex quadratic problem, defined over the hypercube $[0, C]^N$. In general, the solution will have a number of coefficients α_i equal to zero, and since there is a coefficient α_i associated to each data point, only the data points corresponding to non-zero α_i will influence the solution. These points are the support vectors. Intuitively, the support vectors are the data points that lie at the border between the two classes, and a small number of support vectors indicates that the two classes can be well separated.

This technique can be extended to allow for non-linear decision surfaces. This is done by mapping the input vectors into a higher dimensional feature space: $\phi : \Re^q \rightarrow \Re^Q$, and by formulating the linear classification problem in the feature space. Therefore, $f(\mathbf{x})$ can be expressed as $f(\mathbf{x}) = \sum_i \alpha_i y_i \phi^T(\mathbf{x}_i) \cdot \phi(\mathbf{x}) - b$.

If one were given a function $K(\mathbf{x}, \mathbf{y}) = \phi^T(\mathbf{x}) \cdot \phi(\mathbf{y})$, one could learn and use the maximum margin hyperplane in feature space without having to compute explicitly the image of points in $\Re^Q$. It has been proved (Mercer's Theorem) that for each continuous positive definite function $K(\mathbf{x}, \mathbf{y})$ there exists a mapping ϕ such that $K(\mathbf{x}, \mathbf{y}) = \phi^T(\mathbf{x}) \cdot \phi(\mathbf{y})$, $\forall \mathbf{x}, \mathbf{y} \in \Re^q$. By making use of such function K (kernel function), the equation for $f(\mathbf{x})$ can be rewritten as

$$f(\mathbf{x}) = \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) - b. \tag{2.35}$$

Examples of kernel functions are:

$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^p$ (polynomial);

$K(\mathbf{x}, \mathbf{y}) = e^{-||\mathbf{x}-\mathbf{y}||^2/2\sigma^2}$ (Gaussian);

$K(\mathbf{x}, \mathbf{y}) = tanh(\kappa \mathbf{x} \cdot \mathbf{y} - \delta)$ (sigmoid).