Contents

Chapter 4

Nonparametric techniques

4.1 Introduction

In Chap. **??** we treated supervised learning under the assumption that the forms of the underlying density functions were known. Alas, in most pattern recognition applications this assumption is suspect; the common parametric forms rarely fit the densities actually encountered in practice. In particular, all of the classical parametric densities are unimodal (have a single local maximum), whereas many practical problems involve multimodal densities. Further, our hopes are rarely fulfilled that a high-dimensional density might be simply represented as the product of one-dimensional functions. In this chapter we shall examine *nonparametric* procedures that can be used with arbitrary distributions and without the assumption that the forms of the underlying densities are known.

There are several types of nonparametric methods of interest in pattern recognition. One consists of procedures for estimating the density functions $p(\mathbf{x}|\omega_j)$ from sample patterns. If these estimates are satisfactory, they can be substituted for the true densities when designing the classifier. Another consists of procedures for directly estimating the a posteriori probabilities $P(\omega_j|\mathbf{x})$. This is closely related to nonparametric design procedures such as the nearest-neighbor rule, which bypass probability estimation and go directly to decision functions. Finally, there are nonparametric procedures for transforming the feature space in the hope that it may be possible to employ parametric methods in the transformed space. These discriminant analysis methods include the Fisher linear discriminant, which provides an important link between the parametric techniques of Chap. **??** and the adaptive techniques of Chaps. **??** & **??**.

4.2 Density estimation

The basic ideas behind many of the methods of estimating an unknown probability density function are very simple, although rigorous demonstrations that the estimates converge require considerable care. The most fundamental techniques rely on the fact that the probability P that a vector $\mathbf{x}$ will fall in a region $\mathcal{R}$ is given by

$$P = \int_{\mathcal{R}} p(\mathbf{x}')\ d\mathbf{x}'. \tag{1}$$

Thus P is a smoothed or averaged version of the density function $p(\mathbf{x})$, and we can estimate this smoothed value of p by estimating the probability P. Suppose that n samples $\mathbf{x}_1, ..., \mathbf{x}_n$ are drawn independently and identically distributed (i.i.d.) according to the probability law $p(\mathbf{x})$. Clearly, the probability that k of these n fall in $\mathcal{R}$ is given by the binomial law

$$P_k = \binom{n}{k} P^k (1-P)^{n-k}, \tag{2}$$

and the expected value for k is

$$\mathcal{E}[k] = nP. \tag{3}$$

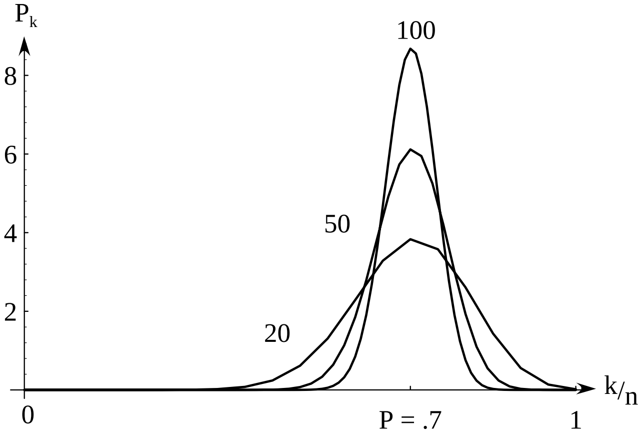

Figure 4.1: The probability P_k of finding k patterns in a volume where the space averaged probability is P as a function of k/n. Each curve is labelled by the total number of patterns n. For large n, such binomial distributions peak strongly at $k/n = P$ (here chosen to be 0.7).

Moreover, this binomial distribution for k peaks very sharply about the mean, so that we expect that the ratio k/n will be a very good estimate for the probability P, and hence for the smoothed density function. This estimate is especially accurate when n is very large (Fig. 4.1). If we now assume that $p(\mathbf{x})$ is continuous and that the region $\mathcal{R}$ is so small that p does not vary appreciably within it, we can write

$$\int_{\mathcal{R}} p(\mathbf{x}')\ d\mathbf{x}' \simeq p(\mathbf{x})V, \tag{4}$$

where $\mathbf{x}$ is a point within $\mathcal{R}$ and V is the volume enclosed by $\mathcal{R}$. Combining Eqs. 1, 3 & 4, we arrive at the following obvious estimate for $p(\mathbf{x})$:

$$p(\mathbf{x}) \simeq \frac{k/n}{V}. \tag{5}$$

There are several problems that remain — some practical and some theoretical. If we fix the volume V and take more and more training samples, the ratio k/n will converge (in probability) as desired, but we have only obtained an estimate of the space-averaged value of $p(\mathbf{x})$,

$$\frac{P}{V} = \frac{\int\limits_{\mathcal{R}} p(\mathbf{x}')\ d\mathbf{x}'}{\int\limits_{\mathcal{R}} d\mathbf{x}'}. \tag{6}$$

If we want to obtain $p(\mathbf{x})$ rather than just an averaged version of it, we must be prepared to let V approach zero. However, if we fix the number n of samples and let V approach zero, the region will eventually become so small that it will enclose no samples, and our estimate $p(\mathbf{x}) \simeq 0$ will be useless. Or if by chance one or more of the training samples coincide at $\mathbf{x}$, the estimate diverges to infinity, which is equally useless.

From a practical standpoint, we note that the number of samples is always limited. Thus, the volume V can not be allowed to become arbitrarily small. If this kind of estimate is to be used, one will have to accept a certain amount of variance in the ratio k/n and a certain amount of averaging of the density $p(\mathbf{x})$.

From a theoretical standpoint, it is interesting to ask how these limitations can be circumvented if an unlimited number of samples is available. Suppose we use the following procedure. To estimate the density at $\mathbf{x}$, we form a sequence of regions $\mathcal{R}_1, \mathcal{R}_2, ...,$ containing $\mathbf{x}$ — the first region to be used with one sample, the second with two, and so on. Let V_n be the volume of $\mathcal{R}_n$, k_n be the number of samples falling in $\mathcal{R}_n$, and $p_n(\mathbf{x})$ be the nth estimate for $p(\mathbf{x})$:

$$p_n(\mathbf{x}) = \frac{k_n/n}{V_n}. \tag{7}$$

If $p_n(\mathbf{x})$ is to converge to $p(\mathbf{x})$, three conditions appear to be required:

- $\lim\limits_{n\to\infty} V_n = 0$
- $\lim\limits_{n\to\infty} k_n = \infty$
- $\lim\limits_{n\to\infty} k_n/n = 0.$

The first condition assures us that the space averaged P/V will converge to $p(\mathbf{x})$, provided that the regions shrink uniformly and that $p(\cdot)$ is continuous at $\mathbf{x}$. The second condition, which only makes sense if $p(\mathbf{x}) \neq 0$, assures us that the frequency ratio will converge (in probability) to the probability P. The third condition is clearly necessary if $p_n(\mathbf{x})$ given by Eq. 7 is to converge at all. It also says that although a huge number of samples will eventually fall within the small region $\mathcal{R}_n$, they will form a negligibly small fraction of the total number of samples.

There are two common ways of obtaining sequences of regions that satisfy these conditions (Fig. 4.2). One is to shrink an initial region by specifying the volume V_n as some function of n, such as $V_n = 1/\sqrt{n}$. It then must be shown that the random variables k_n and k_n/n behave properly, or more to the point, that $p_n(\mathbf{x})$ converges to

$p(\mathbf{x})$. This is basically the Parzen-window method that will be examined in Sect. 4.3. The second method is to specify k_n as some function of n, such as $k_n = \sqrt{n}$. Here the volume V_n is grown until it encloses k_n neighbors of $\mathbf{x}$. This is the k_n-nearest-neighbor estimation method. Both of these methods do in fact converge, although it is difficult to make meaningful statements about their finite-sample behavior.

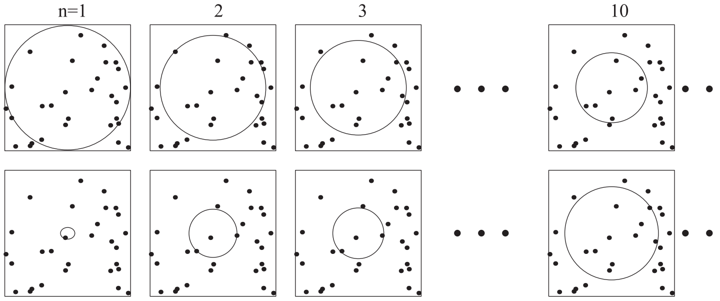

Figure 4.2: Two methods for estimating the density at a point $\mathbf{x}$ (at the center of each square) are to xxx.

4.3 Parzen Windows

The Parzen-window approach to estimating densities can be introduced by temporarily assuming that the region $\mathcal{R}_n$ is a d-dimensional hypercube. If h_n is the length of an edge of that hypercube, then its volume is given by

$$V_n = h_n^d. \tag{8}$$

We can obtain an analytic expression for k_n, the number of samples falling in the hypercube, by defining the following *window function*:

WINDOW FUNCTION

$$\varphi(\mathbf{u}) = \begin{cases} 1 & |u_j| \leq 1/2 \qquad j = 1, ..., d \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

Thus, $\varphi(\mathbf{u})$ defines a unit hypercube centered at the origin. It follows that $\varphi((\mathbf{x} - \mathbf{x}_i)/h_n)$ is equal to unity if $\mathbf{x}_i$ falls within the hypercube of volume V_n centered at $\mathbf{x}$, and is zero otherwise. The number of samples in this hypercube is therefore given by

$$k_n = \sum_{i=1}^{n} \varphi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h_n}\right), \tag{10}$$

and when we substitute this into Eq. 7 we obtain the estimate

$$p_n(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{V_n} \varphi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h_n}\right). \tag{11}$$

This equation suggests a more general approach to estimating density functions. Rather than limiting ourselves to the hypercube window function of Eq. 9, suppose we allow a more general class of window functions. In such a case, Eq. 11 expresses our estimate for $p(\mathbf{x})$ as an average of functions of $\mathbf{x}$ and the samples $\mathbf{x}_i$. In essence, the window function is being used for *interpolation* — each sample contributing to the estimate in accordance with its distance from $\mathbf{x}$.

It is natural to ask that the estimate $p_n(\mathbf{x})$ be a legitimate density function, i.e., that it be nonnegative and integrate to one. This can be assured by requiring the window function itself be a density function. To be more precise, if we require that

$$\varphi(\mathbf{x}) \geq 0 \tag{12}$$

and

$$\int \varphi(\mathbf{u})\; d\mathbf{u} = 1, \tag{13}$$

and if we maintain the relation $V_n = h_n^d$, then it follows at once that $p_n(\mathbf{x})$ also satisfies these conditions.

Let us examine the effect that the *window width* h_n has on $p_n(\mathbf{x})$. If we define the function $\delta_n(\mathbf{x})$ by

$$\delta_n(\mathbf{x}) = \frac{1}{V_n}\varphi\left(\frac{\mathbf{x}}{h_n}\right), \tag{14}$$

then we can write $p_n(\mathbf{x})$ as the average

$$\boxed{p_n(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \delta_n(\mathbf{x} - \mathbf{x}_i).} \tag{15}$$

Since $V_n = h_n^d$, h_n clearly affects both the amplitude and the width of $\delta_n(\mathbf{x})$ (Fig. 4.3). If h_n is very large, the amplitude of δ_n is small, and $\mathbf{x}$ must be far from $\mathbf{x}_i$ before $\delta_n(\mathbf{x} - \mathbf{x}_i)$ changes much from $\delta_n(\mathbf{0})$. In this case, $p_n(\mathbf{x})$ is the superposition of n broad, slowly changing functions and is a very smooth "out-of-focus" estimate of $p(\mathbf{x})$. On the other hand, if h_n is very small, the peak value of $\delta_n(\mathbf{x}-\mathbf{x}_i)$ is large and occurs near $\mathbf{x} = \mathbf{x}_i$. In this case $p(\mathbf{x})$ is the superposition of n sharp pulses centered at the samples — an erratic, "noisy" estimate (Fig. 4.4). For any value of h_n, the distribution is normalized, i.e.,

$$\int \delta_n(\mathbf{x} - \mathbf{x}_i)\; d\mathbf{x} = \int \frac{1}{V_n}\varphi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h_n}\right)\; d\mathbf{x} = \int \varphi(\mathbf{u})\; d\mathbf{u} = 1. \tag{16}$$

Thus, as h_n approaches zero, $\delta_n(\mathbf{x}-\mathbf{x}_i)$ approaches a Dirac delta function centered at $\mathbf{x}_i$, and $p_n(\mathbf{x})$ approaches a superposition of delta functions centered at the samples.

Clearly, the choice of h_n (or V_n) has an important effect on $p_n(\mathbf{x})$. If V_n is too large, the estimate will suffer from too little resolution; if V_n is too small, the estimate will suffer from too much statistical variability. With a limited number of samples, the best we can do is to seek some acceptable compromise. However, with an unlimited number of samples, it is possible to let V_n slowly approach zero as n increases and have $p_n(\mathbf{x})$ converge to the unknown density $p(\mathbf{x})$.

In discussing convergence, we must recognize that we are talking about the convergence of a sequence of random variables, since for any fixed $\mathbf{x}$ the value of $p_n(\mathbf{x})$ depends on the random samples $\mathbf{x}_1, ..., \mathbf{x}_n$. Thus, $p_n(\mathbf{x})$ has some mean $\bar{p}_n(\mathbf{x})$ and variance $\sigma_n^2(\mathbf{x})$. We shall say that the estimate $p_n(\mathbf{x})$ converges to $p(\mathbf{x})$ if

$$\lim_{n\to\infty} \bar{p}_n(\mathbf{x}) = p(\mathbf{x}) \tag{17}$$

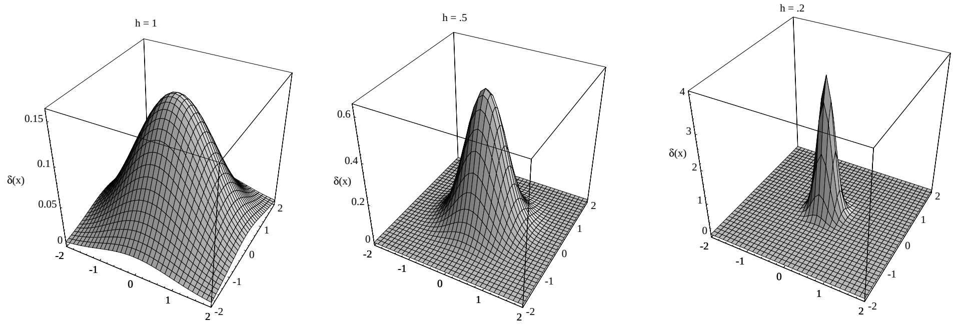

Figure 4.3: Examples of two-dimensional circularly symmetric normal Parzen windows $\varphi(\mathbf{x}/h)$ for three different values of h. Note that because the $\delta_k(\cdot)$ are normalized, different vertical scales must be used to show their structure.

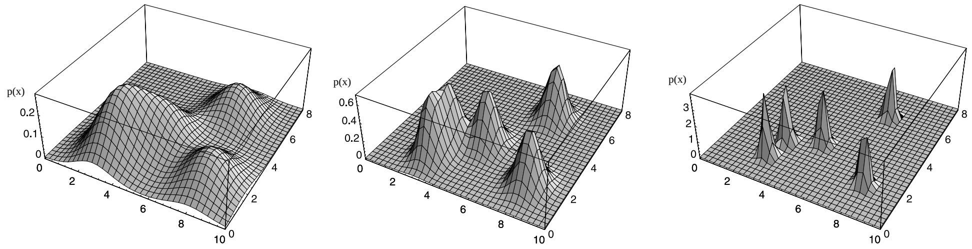

Figure 4.4: Three Parzen-window density estimates based on the same set of five samples, using the window functions in Fig. 4.3. As before, the vertical axes have been scaled to show the structure of each function.

and

$$\lim_{n\to\infty} \sigma_n^2(\mathbf{x}) = 0. \tag{18}$$

To prove convergence we must place conditions on the unknown density $p(\mathbf{x})$, on the window function $\varphi(\mathbf{u})$, and on the window width h_n. In general, continuity of $p(\cdot)$ at $\mathbf{x}$ is required, and the conditions imposed by Eqs. 12 & 13 are customarily invoked. With care, it can be shown that the following additional conditions assure convergence (Problem 1):

$$\sup_{\mathbf{u}} \varphi(\mathbf{u}) < \infty \tag{19}$$

$$\lim_{\|\mathbf{u}\|\to\infty} \varphi(\mathbf{u}) \prod_{i=1}^{d} u_i = 0 \tag{20}$$

$$\lim_{n\to\infty} V_n = 0 \tag{21}$$

and

$$\lim_{n\to\infty} nV_n = \infty. \tag{22}$$

Equations 19 & 20 keep $\varphi(\cdot)$ well behaved, and are satisfied by most density functions that one might think of using for window functions. Equations 21 & 22 state that the volume V_n must approach zero, but at a rate slower than $1/n$. We shall now see why these are the basic conditions for convergence.

4.3.1 Convergence of the Mean

Consider first $\bar{p}_n(\mathbf{x})$, the mean of $p_n(\mathbf{x})$. Since the samples $\mathbf{x}_i$ are i.i.d. according to the (unknown) density $p(\mathbf{x})$, we have

$$\begin{aligned} \bar{p}_n(\mathbf{x}) &= \mathcal{E}[p_n(\mathbf{x})] \\ &= \frac{1}{n}\sum_{i=1}^{n}\mathcal{E}\Big[\frac{1}{V_n}\varphi\Big(\frac{\mathbf{x}-\mathbf{x}_i}{h_n}\Big)\Big] \\ &= \int \frac{1}{V_n}\varphi\Big(\frac{\mathbf{x}-\mathbf{v}}{h_n}\Big)\ p(\mathbf{v})\ d\mathbf{v} \\ &= \int \delta_n(\mathbf{x}-\mathbf{v})p(\mathbf{v})\ d\mathbf{v}. \end{aligned} \tag{23}$$

This equation shows that the expected value of the estimate is an averaged value of the unknown density — a *convolution* of the unknown density and the window function (Appendix **??**). Thus, $\bar{p}_n(\mathbf{x})$ is a blurred version of $p(\mathbf{x})$ as seen through the averaging window. But as V_n approaches zero, $\delta_n(\mathbf{x}-\mathbf{v})$ approaches a delta function centered at $\mathbf{x}$. Thus, if p is continuous at $\mathbf{x}$, Eq. 21 ensures that $\bar{p}_n(\mathbf{x})$ will approach $p(\mathbf{x})$ as n approaches infinity.

CONVOLUTION

4.3.2 Convergence of the Variance

Equation 23 shows that there is no need for an infinite number of samples to make $\bar{p}_n(\mathbf{x})$ approach $p(\mathbf{x})$; one can achieve this for any n merely by letting V_n approach zero. Of course, for a particular set of n samples the resulting "spiky" estimate is useless; this fact highlights the need for us to consider the variance of the estimate. Since $p_n(\mathbf{x})$ is the sum of functions of statistically independent random variables, its variance is the sum of the variances of the separate terms, and hence

$$\begin{aligned} \sigma_n^2(\mathbf{x}) &= \sum_{i=1}^{n}\mathcal{E}\left[\left(\frac{1}{nV_n}\varphi\left(\frac{\mathbf{x}-\mathbf{x}_i}{h_n} - \frac{1}{n}\bar{p}_n(\mathbf{x})\right)\right)^2\right] \\ &= n\ \mathcal{E}\Big[\frac{1}{n^2V_n^2}\varphi^2\Big(\frac{\mathbf{x}-\mathbf{x}_i}{h_n}\Big)\Big] - \frac{1}{n}\bar{p}_n^2(\mathbf{x}) \\ &= \frac{1}{nV_n}\int \frac{1}{V_n}\varphi^2\Big(\frac{\mathbf{x}-\mathbf{v}}{h_n}\Big)\ p(\mathbf{v})\ d\mathbf{v} - \frac{1}{n}\bar{p}_n^2(\mathbf{x}). \end{aligned} \tag{24}$$

By dropping the second term, bounding $\varphi(\cdot)$ and using Eq. 21, we obtain

$$\sigma_n^2(\mathbf{x}) \leq \frac{\sup(\varphi(\cdot))\ \bar{p}_n(\mathbf{x})}{nV_n}. \tag{25}$$

Clearly, to obtain a small variance we want a large value for V_n, not a small one — a large V_n smooths out the local variations in density. However, since the numerator stays finite as n approaches infinity, we can let V_n approach zero and still obtain zero variance, provided that nV_n approaches infinity. For example, we can let $V_n = V_1/\sqrt{n}$ or $V_1/\ln n$ or any other function satisfying Eqs. 21 & 22.

This is the principal theoretical result. Unfortunately, it does not tell us how to choose $\varphi(\cdot)$ and V_n to obtain good results in the finite sample case. Indeed, unless we have more knowledge about $p(\mathbf{x})$ than the mere fact that it is continuous, we have no direct basis for optimizing finite sample results.

4.3.3 Illustrations

It is interesting to see how the Parzen window method behaves on some simple examples, and particularly the effect of the window function. Consider first the case where $p(\mathbf{x})$ is a zero-mean, unit-variance, univariate normal density. Let the window function be of the same form:

$$\varphi(u) = \frac{1}{\sqrt{2\pi}}\ e^{-u^2/2}. \tag{26}$$

Finally, let $h_n = h_1/\sqrt{n}$, where h_1 is a parameter at our disposal. Thus $p_n(x)$ is an average of normal densities centered at the samples:

$$p_n(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h_n}\varphi\Big(\frac{x-x_i}{h_n}\Big). \tag{27}$$

While it is not hard to evaluate Eqs. 23 & 24 to find the mean and variance of $p_n(x)$, it is even more interesting to see numerical results. When a particular set of normally distributed random samples was generated and used to compute $p_n(x)$, the results shown in Fig. 4.5 were obtained. These results depend both on n and h_1. For $n = 1$, $p_n(x)$ is merely a single Gaussian centered about the first sample, which of course has neither the mean nor the variance of the true distribution. For $n = 10$ and $h_1 = 0.1$ the contributions of the individual samples are clearly discernible; this is not the case for $h_1 = 1$ and $h_1 = 5$. As n gets larger, the ability of $p_n(x)$ to resolve variations in $p(x)$ increases. Concomitantly, $p_n(x)$ appears to be more sensitive to local sampling irregularities when n is large, although we are assured that $p_n(x)$ will converge to the smooth normal curve as n goes to infinity. While one should not judge on visual appearance alone, it is clear that many samples are required to obtain an accurate estimate. Figure 4.6 shows analogous results in two dimensions.

As a second one-dimensional example, we let $\varphi(x)$ and h_n be the same as in Fig. 4.5, but let the unknown density be a mixture of two uniform densities:

$$p(x) = \begin{cases} 1 & -2.5 < x < -2 \\ 1/4 & 0 < x < 2 \\ 0 & \text{otherwise.} \end{cases} \tag{28}$$

Figure 4.7 shows the behavior of Parzen-window estimates for this density. As before, the case $n = 1$ tells more about the window function than it tells about the unknown

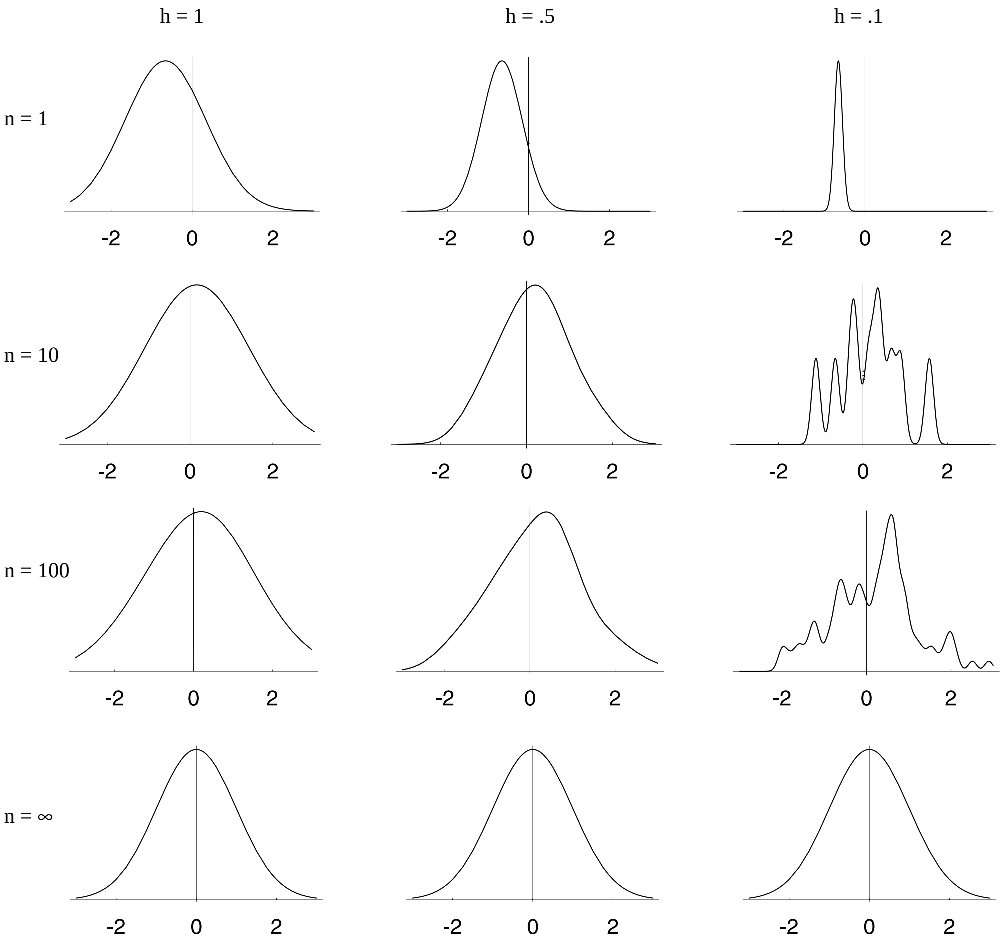

Figure 4.5: Parzen-window estimates of a univariate normal density using different window widths and numbers of samples. The vertical axes have been scaled to best show the structure in each graph. Note particularly that the $n = \infty$ estimates are the same (and match the true generating function), regardless of window width h.

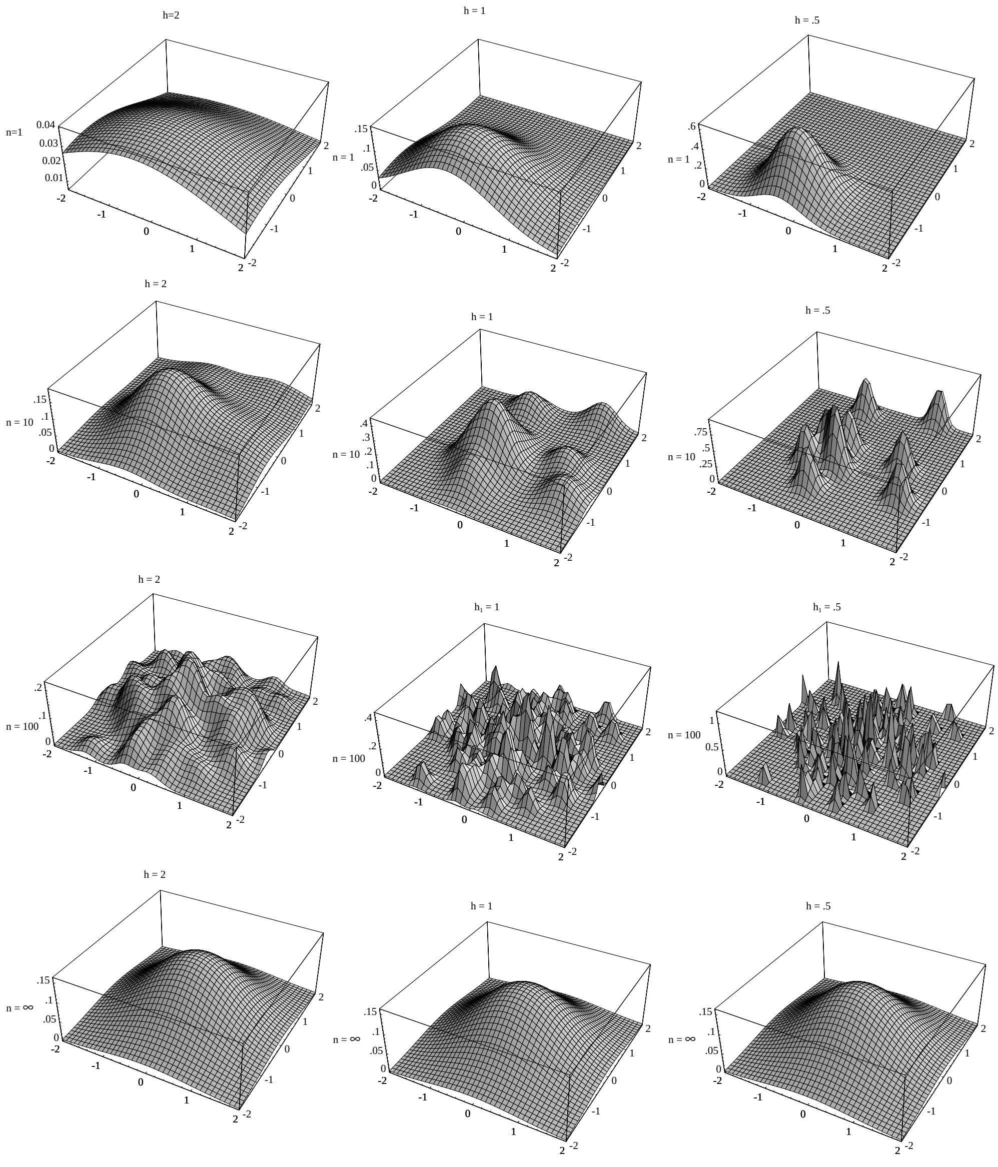

Figure 4.6: Parzen-window estimates of a bivariate normal density using different window widths and numbers of samples. The vertical axes have been scaled to best show the structure in each graph. Note particularly that the $n = \infty$ estimates are the same (and match the true generating distribution), regardless of window width h.

density. For $n = 16$, none of the estimates is particularly good, but results for $n = 256$ and $h_1 = 1$ are beginning to appear acceptable.

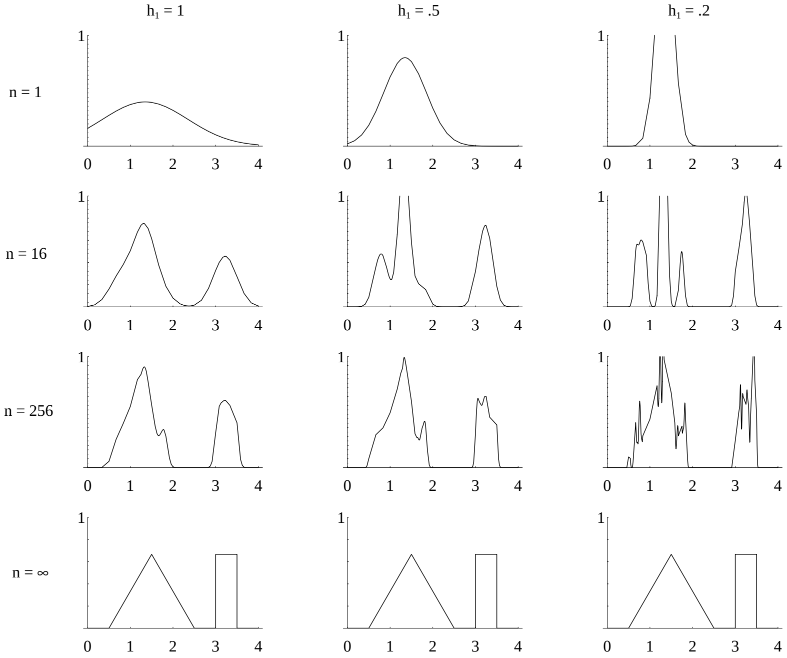

Figure 4.7: Parzen-window estimates of a bimodal distribution using different window widths and numbers of samples. Note particularly that the $n = \infty$ estimates are the same (and match the true generating distribution), regardless of window width h.

4.3.4 Classification example

In classifiers based on Parzen-window estimation, we estimate the densities for each category and classify a test point by the label corresponding to the maximum posterior. If there are multiple categories with unequal priors we can easily include these too (Problem 4). The decision regions for a Parzen-window classifier depend upon the choice of window function, of course, as illustrated in Fig. 4.8. In general, the training error — the empirical error on the training points themselves — can be made arbitrarily low by making the window width sufficiently small.* However, the goal of creating a classifier is to classify novel patterns, and alas a low training error does not guarantee a small *test* error, as we shall explore in Chap. **??**. Although a generic Gaussian window *shape* can be justified by considerations of noise, statistical independence and uncertainty, in the absense of other information about the underlying distributions there is little theoretical justification of one window *width* over another.

These density estimation and classification examples illustrate some of the power and some of the limitations of nonparametric methods. Their power resides in their generality. Exactly the same procedure was used for the unimodal normal case and the bimodal mixture case and we did not need to make any assumptions about the

* We ignore cases in which the same feature vector has been assigned to multiple categories.

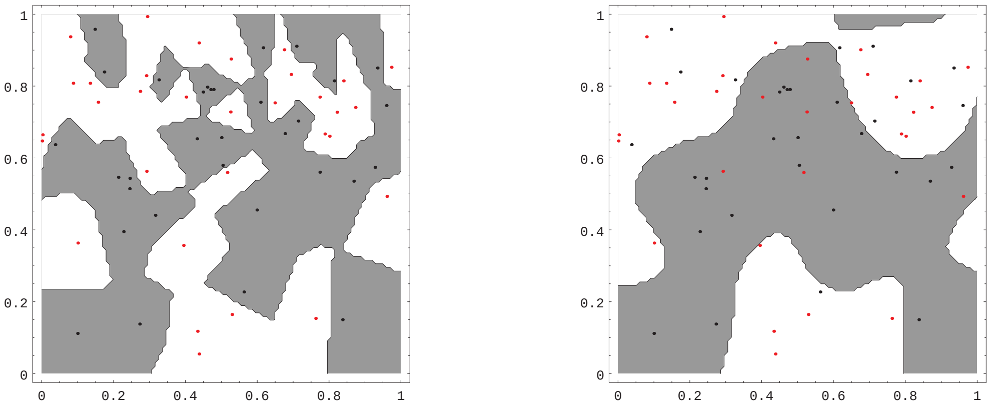

Figure 4.8: The decision boundaries in a two-dimensional Parzen-window dichotomizer depend on the window width h. At the left a small h leads to boundaries that are more complicated than for large h on same data set, shown at the right. Apparently, for this data a small h would be appropriate for the upper region, while a large h for the lower region; no single window width is ideal overall.

distributions ahead of time. With enough samples, we are essentially assured of convergence to an arbitrarily complicated target density. On the other hand, the number of samples needed may be very large indeed — much greater than would be required if we knew the form of the unknown density. Little or nothing in the way of data reduction is provided, which leads to severe requirements for computation time and storage. Moreover, the demand for a large number of samples grows exponentially with the dimensionality of the feature space. This limitation is related to the "curse of dimensionality," and severely restricts the practical application of such nonparametric procedures (Problem 11). The fundamental reason for the curse of dimensionality is that high-dimensional functions have the potential to be much more complicated than low-dimensional ones, and that those complications are harder to discern. The only way to beat the curse is to incorporate knowledge about the data that is *correct*.

4.3.5 Probabilistic Neural Networks (PNNs)

A hardware implementation of the Parzen windows approach is found in Probabilistic Neural Networks (Fig. 4.9). Suppose we wish to form a Parzen estimate based on n patterns, each of which is d-dimensional, randomly sampled from c classes. The PNN for this case consists of d *input units* comprising the input layer, each unit is connect to each of the n *pattern units*; each pattern unit is, in turn, connected to one and only one of the c category units. The connections from the input to pattern units represent modifiable *weights*, which will be trained. (While these weights are merely parameters and could be represented by a vector $\boldsymbol{\theta}$, in keeping with the established terminology in neural networks we shall use the symbol $\mathbf{w}$.) Each link from a pattern unit to its associated category unit is of a single constant magnitude.

INPUT UNIT

PATTERN UNIT

CATEGORY UNIT

WEIGHT

The PNN is trained in the following way. First, each pattern $\mathbf{x}$ of the training set is

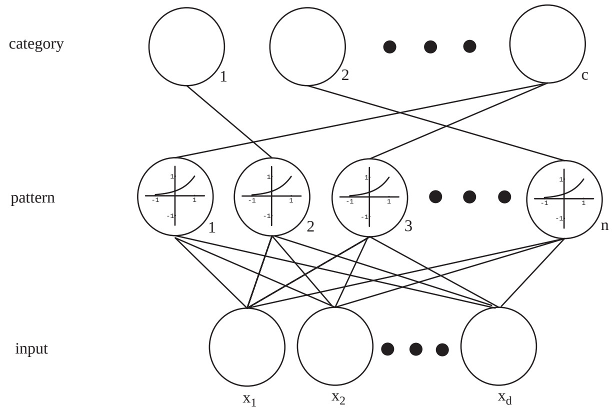

Figure 4.9: A probabilistic neural network (PNN) consists of d input units, n pattern units and c category units. Each pattern unit forms the inner product of its weight vector and the normalized pattern vector $\mathbf{x}$ to form $z = \mathbf{w}^t\mathbf{x}$, and then emits $\exp[(z-1)/\sigma^2]$. Each category unit sums such contributions from the pattern unit connected to it. This insures that the activity in each of the category units represents the Parzen-window density estimate using a circularly symmetric Gaussian window of covariance $\sigma^2\mathbf{I}$, where $\mathbf{I}$ is the $d \times d$ identity matrix.

normalized to have unit length, i.e., is scaled so that $\sum\limits_{i=1}^{d} x_i^2 = 1$.[*] The first normalized training pattern is placed on the input units. The modifiable weights linking the input units and the first pattern unit are set such that $\mathbf{w}_1 = \mathbf{x}_1$. (Note that because of the normalization of $\mathbf{x}_1$, $\mathbf{w}_1$ is normalized too.) Then, a single connection from the first pattern unit is made to the category unit corresponding to the known class of that pattern. The process is repeated with each of the remaining training patterns, setting the weights to the successive pattern units such that $\mathbf{w}_k = \mathbf{x}_k$ for $k = 1, 2, ..., n$. After such training we have a network that is fully connected between input and pattern units, and sparsely connected from pattern to category units. If we denote the components of the jth pattern as x_{jk} and the weights to the jth pattern unit w_{jk}, for $j = 1, 2, ..., n$ and $k = 1, 2, ..., d$, then our algorithm is:

Algorithm 1 (PNN training)

```
begin initialize $j = 0, n = \#\text{patterns}$
        do $j \leftarrow j + 1$
            normalize : $x_{jk} \leftarrow x_{jk} / \left(\sum_i^d x_{ji}^2\right)^{1/2}$
            train : $w_{jk} \leftarrow x_{jk}$
            if $\mathbf{x} \in \omega_i$ then $a_{ic} \leftarrow 1$
        until $j = n$
```

[*] Such normalization collapses two vectors having the same direction but different magnitude. In order to avoid this, we can augment the pattern with a feature of magnitude 1.0, making it $(d+1)$-dimensional, and then normalize.

```
end
```

The trained network is then used for classification in the following way. A normalized test pattern $\mathbf{x}$ is placed at the input units. Each pattern unit computes the inner product

$$z_k = \mathbf{w}_k^t \mathbf{x}, \tag{29}$$

and emits a nonlinear function of z_k; each output unit sums the contributions from all pattern units connected to it. The nonlinear function is $e^{(z_k-1)/\sigma^2}$, where σ is a parameter set by the user and is equal to $\sqrt{2}$ times the width of the effective Gaussian window. To understand this choice of nonlinearity, consider an (unnormalized) Gaussian window centered on the position of one of the training patterns $\mathbf{w}_k$. We work backwards from the desired Gaussian window function to infer the nonlinear transfer function that should be employed by the pattern units. That is, if we let our effective width h_n be a constant, the window function is

$$\begin{aligned}\varphi\left(\frac{\mathbf{x}_k - \mathbf{w}_k}{h_n}\right) &\propto \overbrace{e^{-(\mathbf{x}-\mathbf{w}_k)^t(\mathbf{x}-\mathbf{w}_k)/2\sigma^2}}^{\text{desired Gaussian}} \\ &= e^{-(\mathbf{x}^t\mathbf{x}+\mathbf{w}_k^t\mathbf{w}_k-2\mathbf{x}^t\mathbf{w}_k)/2\sigma^2} = \underbrace{e^{(z_k-1)/\sigma^2}}_{\substack{\text{transfer}\\ \text{function}}},\end{aligned} \tag{30}$$

where we have used our normalization conditions $\mathbf{x}^t\mathbf{x} = \mathbf{w}_k^t\mathbf{w}_k = 1$. Thus each pattern unit contributes to its associated category unit a signal equal to the probability the test point was generated by a Gaussian centered on the associated training point. The sum of these local estimates (computed at the corresponding category unit) gives the discriminant function $g_i(\mathbf{x})$ — the Parzen window estimate of the underlying distribution. The $\max_i g_i(\mathbf{x})$ operation gives the desired category for the test point (Algorithm 2).

Algorithm 2 (PNN classification)

```
begin initialize $k = 0, \mathbf{x} =$ test pattern
          do $k \leftarrow k + 1$
              $z_k \leftarrow \mathbf{w}_k^t \mathbf{x}$
              if $a_{kc} = 1$ then  $g_c \leftarrow g_c + \exp[(z_k - 1)/\sigma^2]$
          until $k = n$
  return $class \leftarrow \arg\max_i g_i(\mathbf{x})$
end
```

One of the benefits of PNNs is their speed of learning, since the learning rule (i.e., setting $\mathbf{w}_k = \mathbf{x}_k$) is simple and requires only a single pass through the training data. The space complexity (amount of memory) for the PNN is easy to determine by counting the number of wires in Fig. 4.9 — $O((n+1)d)$. This can be quite severe for instance in a hardware application, since both n and d can be quite large. The time complexity for classification by the parallel implementation of Fig. 4.9 is $O(1)$, since the n inner products of Eq. 29 can be done in parallel. Thus this PNN architecture could find uses where recognition speed is important and storage is not a severe limitation. Another benefit is that new training patterns can be incorporated into a previously trained classifier quite easily; this might be important for a particular on-line application.

4.3.6 Choosing the window function

As we have seen, one of the problems encountered in the Parzen-window/PNN approach concerns the choice of the sequence of cell volumes sizes $V_1, V_2, ...$ or overall window size (or indeed other window parameters, such as shape or orientation). For example, if we take $V_n = V_1/\sqrt{n}$, the results for any finite n will be very sensitive to the choice for the initial volume V_1. If V_1 is too small, most of the volumes will be empty, and the estimate $p_n(\mathbf{x})$ will be very erratic (Fig. 4.7). On the other hand, if V_1 is too large, important spatial variations in $p(\mathbf{x})$ may be lost due to averaging over the cell volume. Furthermore, it may well be the case that a cell volume appropriate for one region of the feature space might be entirely unsuitable in a different region (Fig. 4.8). In Chap. **??** we shall consider general methods, including cross-validation, which are often used in conjunction with Parzen windows. Now, though, we turn to an important alternative method that is both useful and has solvable analytic properties.

4.4 k_n–Nearest-Neighbor Estimation

A potential remedy for the problem of the unknown "best" window function is to let the cell volume be a function of the *training data*, rather than some arbitrary function of the overall number of samples. For example, to estimate $p(\mathbf{x})$ from n training samples or *prototypes* we can center a cell about $\mathbf{x}$ and let it grow until it captures k_n samples, where k_n is some specified function of n. These samples are the k_n *nearest-neighbors* of $\mathbf{x}$. It the density is high near $\mathbf{x}$, the cell will be relatively small, which leads to good resolution. If the density is low, it is true that the cell will grow large, but it will stop soon after it enters regions of higher density. In either case, if we take

PROTOTYPES

$$p_n(\mathbf{x}) = \frac{k_n/n}{V_n} \tag{31}$$

we want k_n to go to infinity as n goes to infinity, since this assures us that k_n/n will be a good estimate of the probability that a point will fall in the cell of volume V_n. However, we also want k_n to grow sufficiently slowly that the size of the cell needed to capture k_n training samples will shrink to zero. Thus, it is clear from Eq. 31 that the ratio k_n/n must go to zero. Although we shall not supply a proof, it can be shown that the conditions $\lim\limits_{n\to\infty} k_n = \infty$ and $\lim\limits_{n\to\infty} k_n/n = 0$ are necessary and sufficient for $p_n(\mathbf{x})$ to converge to $p(\mathbf{x})$ in probability at all points where $p(\mathbf{x})$ is continuous (Problem 5). If we take $k_n = \sqrt{n}$ and assume that $p_n(\mathbf{x})$ is a reasonably good approximation to $p(\mathbf{x})$ we then see from Eq. 31 that $V_n \simeq 1/(\sqrt{n}p(\mathbf{x}))$. Thus, V_n again has the form $V_1/\sqrt{n}$, but the initial volume V_1 is determined by the nature of the data rather than by some arbitrary choice on our part. Note that there are nearly always discontinuities in the slopes of these estimates, and these lie away from the prototypes themselves (Figs. 4.10 & 4.11).

It is instructive to compare the performance of this method with that of the Parzen-window/PNN method on the data used in the previous examples. With $n = 1$

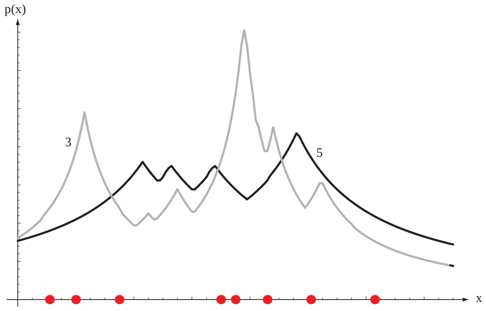

Figure 4.10: Eight points in one dimension and the k-nearest-neighbor density estimates, for $k = 3$ and 5. Note especially that the discontinuities in the slopes in the estimates generally occur *away* fom the positions of the points themselves.

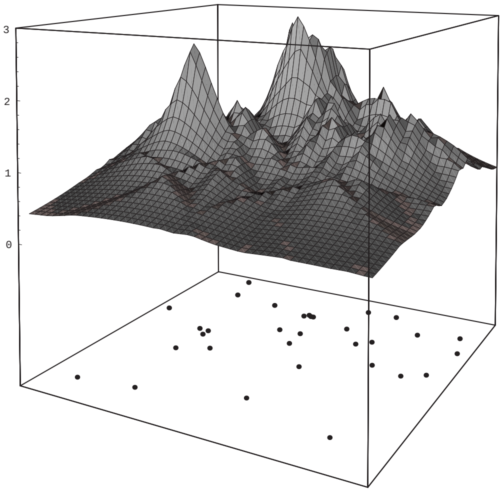

Figure 4.11: The k-nearest-neighbor estimate of a two-dimensional density for $k = 5$. Notice how such a finite n estimate can be quite "jagged," and that discontinuities in the slopes generally occur along lines away from the positions of the points themselves.

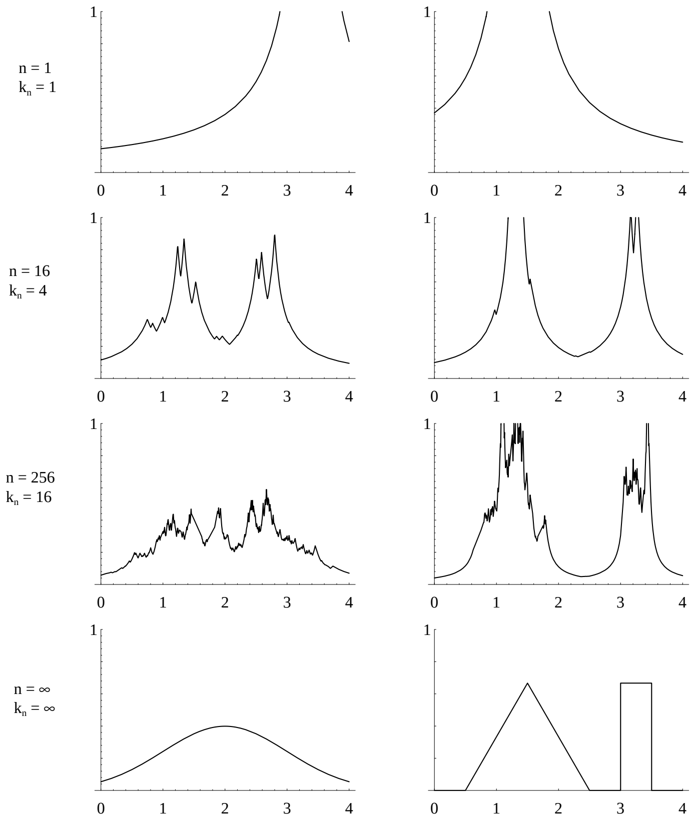

Figure 4.12: Several k-nearest-neighbor estimates of two unidimensional densities: a Gaussian and a bimodal distribution. Notice how the finite n estimates can be quite "spiky."

and $k_n = \sqrt{n} = 1$, the estimate becomes

$$p_n(x) = \frac{1}{2|x - x_1|}. \tag{32}$$

This is clearly a poor estimate of $p(x)$, with its integral embarrassing us by diverging to infinity. As shown in Fig. 4.12, the estimate becomes considerably better as n gets larger, even though the integral of the estimate remains infinite. This unfortunate fact is compensated by the fact that $p_n(x)$ never plunges to zero just because no samples fall within some arbitrary cell or window. While this might seem to be a meager compensation, it can be of considerable value in higher-dimensional spaces.

As with the Parzen-window approach, we could obtain a family of estimates by taking $k_n = k_1\sqrt{n}$ and choosing different values for k_1. However, in the absense of any additional information, one choice is as good as another, and we can be confident only that the results will be correct in the infinite data case. For classification, one popular method is to adjust the window width until the classifier has the lowest error on a separate set of samples, also drawn from the target distributions, a technique we shall explore in Chap. **??**.

4.4.1 Estimation of a posteriori probabilities

The techniques discussed in the previous sections can be used to estimate the a posteriori probabilities $P(\omega_i|\mathbf{x})$ from a set of n labelled samples by using the samples to estimate the densities involved. Suppose that we place a cell of volume V around $\mathbf{x}$ and capture k samples, k_i of which turn out to be labelled ω_i. Then the obvious estimate for the joint probability $p(\mathbf{x},\omega_i)$ is

$$p_n(\mathbf{x},\omega_i) = \frac{k_i/n}{V}, \tag{33}$$

and thus a reasonable estimate for $P(\omega_i|\mathbf{x})$ is

$$P_n(\omega_i|\mathbf{x}) = \frac{p_n(\mathbf{x},\omega_i)}{\sum\limits_{j=1}^{c} p_n(\mathbf{x},\omega_j)} = \frac{k_i}{k}. \tag{34}$$

That is, the estimate of the a posteriori probability that ω_i is the state of nature is merely the fraction of the samples within the cell that are labelled ω_i. Consequently, for minimum error rate we select the category most frequently represented within the cell. If there are enough samples and if the cell is sufficiently small, it can be shown that this will yield performance approaching the best possible.

When it comes to choosing the size of the cell, it is clear that we can use either the Parzen-window approach or the k_n-nearest-neighbor approach. In the first case, V_n would be some specified function of n, such as $V_n = 1/\sqrt{n}$. In the second case, V_n would be expanded until some specified number of samples were captured, such as $k = \sqrt{n}$. In either case, as n goes to infinity an infinite number of samples will fall within the infinitely small cell. The fact that the cell volume could become arbitrarily small and yet contain an arbitrarily large number of samples would allow us to learn the unknown probabilities with virtual certainty and thus eventually obtain optimum performance. Interestingly enough, we shall now see that we can obtain comparable performance if we base our decison solely on the label of the *single* nearest neighbor of $\mathbf{x}$.

4.5 The Nearest-Neighbor Rule

While the k-nearest-neighbor algorithm was first proposed for arbitrary k, the crucial matter of determining the error bound was first solved for $k = 1$. This nearest-neighbor algorithm has conceptual and computational simplicity. We begin by letting $\mathcal{D}^n = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ denote a set of n labelled prototypes, and $\mathbf{x}' \in \mathcal{D}^n$ be the prototype nearest to a test point $\mathbf{x}$. Then the *nearest-neighbor rule* for classifying $\mathbf{x}$ is to assign it the label associated with $\mathbf{x}'$. The nearest-neighbor rule is a sub-optimal procedure; its use will usually lead to an error rate greater than the minimum possible, the Bayes rate. We shall see, however, that with an unlimited number of prototypes the error rate is never worse than twice the Bayes rate.

Before we get immersed in details, let us try to gain a heuristic understanding of why the nearest-neighbor rule should work so well. To begin with, note that the label θ' associated with the nearest neighbor is a random variable, and the probability that $\theta' = \omega_i$ is merely the a posteriori probability $P(\omega_i|\mathbf{x}')$. When the number of samples is very large, it is reasonable to assume that $\mathbf{x}'$ is sufficiently close to $\mathbf{x}$ that $P(\omega|\mathbf{x}') \simeq P(\omega_i|\mathbf{x})$. Since this is exactly the probability that nature will be in state ω_i, the nearest-neighbor rule is effectively matching probabilities with nature.

If we define $\omega_m(\mathbf{x})$ by

$$P(\omega_m|\mathbf{x}) = \max_i P(\omega_i|\mathbf{x}), \tag{35}$$

then the Bayes decision rule always selects ω_m. This rule allows us to partition the feature space into cells consisting of all points closer to a given training point $\mathbf{x}'$ than to any other training points. All points in such a cell are thus labelled by the category of the training point — a so-called *Voronoi tesselation* of the space (Fig. 4.13).

VORONOI TESSELATION

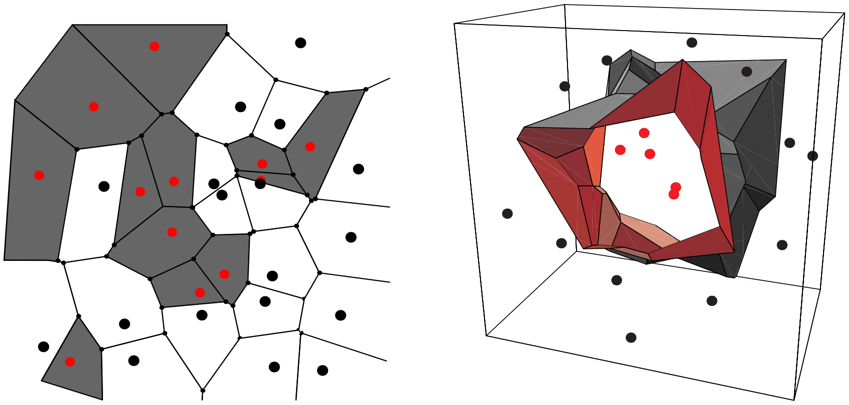

Figure 4.13: In two dimensions, the nearest-neighbor algorithm leads to a partitioning of the input space into Voronoi cells, each labelled by the category of the training point it contains. In three dimensions, the cells are three-dimensional, and the decision boundary resembles the surface of a crystal.

When $P(\omega_m|\mathbf{x})$ is close to unity, 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 $P(\omega_m|\mathbf{x})$ is close to $1/c$, so that all classes are essentially equally likely, the selections made by the nearest-neighbor rule and the Bayes decision rule are rarely the same, but the probability of error is approximately $1 - 1/c$ for both. While more careful analysis is clearly necessary, these observations should make the good performance of the nearest-neighbor rule less surprising.

Our analysis of the behavior of the nearest-neighbor rule will be directed at obtaining the infinite-sample conditional average probability of error $P(e|\mathbf{x})$, where the averaging is with respect to the training samples. The unconditional average probability of error will then be found by averaging $P(e|\mathbf{x})$ over all $\mathbf{x}$:

$$P(e) = \int P(e|\mathbf{x})p(\mathbf{x})\, d\mathbf{x}. \tag{36}$$

In passing we should recall that the Bayes decision rule minimizes $P(e)$ by minimizing $P(e|\mathbf{x})$ for every $\mathbf{x}$. Recall from Chap. **??** that if we let $P^*(e|\mathbf{x})$ be the minimum possible value of $P(e|\mathbf{x})$, and P^* be the minimum possible value of $P(e)$, then

$$P^*(e|\mathbf{x}) = 1 - P(\omega_m|\mathbf{x}) \tag{37}$$

and

$$P^* = \int P^*(e|\mathbf{x})p(\mathbf{x})\ d\mathbf{x}. \tag{38}$$

4.5.1 Convergence of the Nearest Neighbor

We now wish to evaluate the average probability of error for the nearest-neighbor rule. In particular, if $P_n(e)$ is the n-sample error rate, and if

$$P = \lim_{n\to\infty} P_n(e), \tag{39}$$

then we want to show that

$$P^* \leq P \leq P^*\Big(2 - \frac{c}{c-1}P^*\Big). \tag{40}$$

We begin by observing that when the nearest-neighbor rule is used with a particular set of n samples, the resulting error rate will depend on the accidental characteristics of the samples. In particular, if different sets of n samples are used to classify $\mathbf{x}$, different vectors $\mathbf{x}'$ will be obtained for the nearest-neighbor of $\mathbf{x}$. Since the decision rule depends on this nearest-neighbor, we have a conditional probability of error $P(e|\mathbf{x}, \mathbf{x}')$ that depends on both $\mathbf{x}$ and $\mathbf{x}'$. By averaging over $\mathbf{x}'$, we obtain

$$P(e|\mathbf{x}) = \int P(e|\mathbf{x}, \mathbf{x}')p(\mathbf{x}'|\mathbf{x})\ d\mathbf{x}'. \tag{41}$$

where we understand that there is an implicit dependence upon the number n of training points.

It is usually very difficult to obtain an exact expression for the conditional density $p(\mathbf{x}'|\mathbf{x})$. However, since $\mathbf{x}'$ is by definition the nearest-neighbor of $\mathbf{x}$, we expect this density to be very peaked in the immediate vicinity of $\mathbf{x}$, and very small elsewhere. Furthermore, as n goes to infinity we expect $p(\mathbf{x}'|\mathbf{x})$ to approach a delta function centered at $\mathbf{x}$, making the evaluation of Eq. 41 trivial. To show that this is indeed the case, we must assume that at the given $\mathbf{x}$, $p(\cdot)$ is continuous and not equal to zero. Under these conditions, the probability that any sample falls within a hypersphere $\mathcal{S}$ centered about $\mathbf{x}$ is some positive number P_s:

$$P_s = \int\limits_{\mathbf{x}'\in\mathcal{S}} p(\mathbf{x}')\ d\mathbf{x}'. \tag{42}$$

Thus, the probability that all n of the independently drawn samples fall outside this hypersphere is $(1 - P_s)^n$, which approaches zero as n goes to infinity. Thus $\mathbf{x}'$ converges to $\mathbf{x}$ in probability, and $p(\mathbf{x}'|\mathbf{x})$ approaches a delta function, as expected. In fact, by using measure theoretic methods one can make even stronger (as well as more rigorous) statements about the convergence of $\mathbf{x}'$ to $\mathbf{x}$, but this result is sufficient for our purposes.

4.5.2 Error Rate for the Nearest-Neighbor Rule

We now turn to the calculation of the conditional probability of error $P_n(e|\mathbf{x}, \mathbf{x}')$. To avoid a potential source of confusion, we must state the problem with somewhat greater care than has been exercised so far. When we say that we have n independently drawn labelled samples, we are talking about n pairs of random variables $(\mathbf{x}_1, \theta_1), (\mathbf{x}_2, \theta_2), ..., (\mathbf{x}_n, \theta_n)$, where θ_j may be any of the c states of nature $\omega_1, ..., \omega_c$. We assume that these pairs were generated by selecting a state of nature ω_j for θ_j with probability $P(\omega_j)$ and then selecting an $\mathbf{x}_j$ according to the probability law $p(\mathbf{x}|\omega_j)$, with each pair being selected independently. Suppose that during classification nature selects a pair $(\mathbf{x}, \theta)$, and that $\mathbf{x}'_j$, labelled θ'_j, is the training sample nearest $\mathbf{x}$. Since the state of nature when $\mathbf{x}'_j$ was drawn is independent of the state of nature when $\mathbf{x}$ is drawn, we have

$$P(\theta, \theta'_j|\mathbf{x}, \mathbf{x}'_j) = P(\theta|\mathbf{x})P(\theta'_j|\mathbf{x}'_j). \tag{43}$$

Now if we use the nearest-neighbor decision rule, we commit an error whenever $\theta \neq \theta'_j$. Thus, the conditional probability of error $P_n(e|\mathbf{x}, \mathbf{x}'_j)$ is given by

$$\begin{aligned} P_n(e|\mathbf{x}, \mathbf{x}'_j) &= 1 - \sum_{i=1}^{c} P(\theta = \omega_i, \theta' = \omega_i|\mathbf{x}, \mathbf{x}'_j) \\ &= 1 - \sum_{i=1}^{c} P(\omega_i|\mathbf{x})P(\omega_i|\mathbf{x}'_j). \end{aligned} \tag{44}$$

To obtain $P_n(e)$ we must substitute this expression into Eq. 41 for $P_n(e|\mathbf{x})$ and then average the result over $\mathbf{x}$. This is very difficult, in general, but as we remarked earlier the integration called for in Eq. 41 becomes trivial as n goes to infinity and $p(\mathbf{x}'|\mathbf{x})$ approaches a delta function. If $P(\omega_i|\mathbf{x})$ is continuous at $\mathbf{x}$, we thus obtain

$$\begin{aligned} \lim_{n\to\infty} P_n(e|\mathbf{x}) &= \int \Big[1 - \sum_{i=1}^{c} P(\omega_i|\mathbf{x})P(\omega_i|\mathbf{x}')\Big]\delta(\mathbf{x}' - \mathbf{x})\ d\mathbf{x}' \\ &= 1 - \sum_{i=1}^{c} P^2(\omega_i|\mathbf{x}). \end{aligned} \tag{45}$$

Therefore, provided we can exchange some limits and integrals, the asymptotic nearest-neighbor error rate is given by

$$\begin{aligned} P &= \lim_{n\to\infty} P_n(e) \\ &= \lim_{n\to\infty} \int P_n(e|\mathbf{x})p(\mathbf{x})\ d\mathbf{x} \\ &= \int \Big[1 - \sum_{i=1}^{c} P^2(\omega_i|\mathbf{x})\Big]p(\mathbf{x})\ d\mathbf{x}. \end{aligned} \tag{46}$$

4.5.3 Error Bounds

While Eq. 46 presents an exact result, it is more illuminating to obtain bounds on P in terms of the Bayes rate P^*. An obvious lower bound on P is P^* itself. Furthermore, it can be shown that for any P^* there is a set of conditional and prior probabilities for which the bound is achieved, so in this sense it is a tight lower bound.

The problem of establishing a tight upper bound is more interesting. The basis for hoping for a low upper bound comes from observing that if the Bayes rate is low, $P(\omega_i|\mathbf{x})$ is near 1.0 for some i, say $i = m$. Thus the integrand in Eq. 46 is approximately $1 - P^2(\omega_m|\mathbf{x}) \simeq 2(1 - P(\omega_m|\mathbf{x}))$, and since

$$P^*(e|\mathbf{x}) = 1 - P(\omega_m|\mathbf{x}), \tag{47}$$

integration over $\mathbf{x}$ might yield about twice the Bayes rate, which is still low and acceptable for some applications. To obtain an exact upper bound, we must find out how large the nearest-neighbor error rate P can become for a given Bayes rate P^*. Thus, Eq. 46 leads us to ask how small $\sum\limits_{i=1}^{c} P^2(\omega_i|\mathbf{x})$ can be for a given $P(\omega_m|\mathbf{x})$. First we write

$$\sum_{i=1}^{c} P^2(\omega_i|\mathbf{x}) = P^2(\omega_m|\mathbf{x}) + \sum_{i \neq m} P^2(\omega_i|\mathbf{x}), \tag{48}$$

and then seek to bound this sum by minimizing the second term subject to the following constraints:

- $P(\omega_i|\mathbf{x}) \geq 0$
- $\sum\limits_{i \neq m} P(\omega_i|\mathbf{x}) = 1 - P(\omega_m|\mathbf{x}) = P^*(e|\mathbf{x})$.

With a little thought we see that $\sum\limits_{i=1}^{c} P^2(\omega_i|\mathbf{x})$ is minimized if all of the a posteriori probabilities except the mth are equal. The second constraint yields

$$P(\omega_i|\mathbf{x}) = \begin{cases} \frac{P^*(e|\mathbf{x})}{c-a} & i \neq m \\ 1 - P^*(e|\mathbf{x}) & i = m. \end{cases} \tag{49}$$

Thus we have the inequalities

$$\sum_{i=1}^{c} P^2(\omega_i|\mathbf{x}) \geq (1 - P^*(e|\mathbf{x}))^2 + \frac{P^{*2}(e|\mathbf{x})}{c-1} \tag{50}$$

and

$$1 - \sum_{i=1}^{c} P^2(\omega_i|\mathbf{x}) \leq 2P^*(e|\mathbf{x}) - \frac{c}{c-1} P^{*2}(e|\mathbf{x}). \tag{51}$$

This immediately shows that $P \leq 2P^*$, since we can substitute this result in Eq. 46 and merely drop the second term. However, a tighter bound can be obtained by observing that the variance is:

$$\begin{aligned} \mathrm{Var}[P^*(e|\mathbf{x})] &= \int [P^*(e|\mathbf{x}) - P^*]^2 p(\mathbf{x})\ d\mathbf{x} \\ &= \int P^{*2}(e|\mathbf{x}) p(\mathbf{x})\ d\mathbf{x} - P^{*2} \geq 0, \end{aligned}$$

so that

$$\int P^{*2}(e|\mathbf{x})p(\mathbf{x})\;d\mathbf{x}\;\geq P^{*2}, \tag{52}$$

with equality holding if and only if the variance of $P^*(e|\mathbf{x})$ is zero. Using this result and substituting Eq. 51 into Eq. 46, we obtain the desired bounds on the nearest-neighbor error P in the case of an infinite number of samples:

$$\boxed{P^* \leq P \leq P^*\Big(2 - \frac{c}{c-1}P^*\Big).} \tag{53}$$

It is easy to show that this upper bound is achieved in the so-called zero-information case in which the densities $p(\mathbf{x}|\omega_i)$ are identical, so that $P(\omega_i|\mathbf{x}) = P(\omega_i)$ and furthermore $P^*(e|\mathbf{x})$ is independent of $\mathbf{x}$ (Problem 17). Thus the bounds given by Eq. 53 are as tight as possible, in the sense that for any P^* there exist conditional and a priori probabilities for which the bounds are achieved. In particular, the Bayes rate P^* can be anywhere between 0 and $(c-1)/c$ and the bounds meet at the two extreme values for the probabilities. When the Bayes rate is small, the upper bound is approximately twice the Bayes rate (Fig. 4.14).

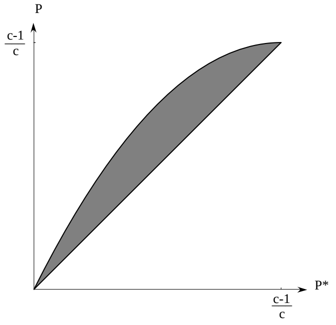

Figure 4.14: Bounds on the nearest-neighbor error rate P in a c-category problem given infinite training data, where P^* is the Bayes error (Eq. 53). At low error rates, the nearest-neighbor error rate is bounded above by twice the Bayes rate.

Since P is always less than or equal to $2P^*$, if one had an infinite collection of data and used an arbitrarily complicated decision rule, one could at most cut the error rate in half. In this sense, at least half of the classification information in an infinite data set resides in the nearest neighbor.

It is natural to ask how well the nearest-neighbor rule works in the finite-sample case, and how rapidly the performance converges to the asymptotic value. Unfortunately, despite prolonged effort on such problems, 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 $P_n(e)$ need not even decrease monotonically with n. As with other nonparametric methods, it is difficult to obtain anything other than asymptotic results without making further assumptions about the underlying probability structure (Problems 13 & 14).

4.5.4 The k-Nearest-Neighbor Rule

An obvious extension of the nearest-neighbor rule is the *k-nearest-neighbor rule*. As one would expect from the name, this rule classifies $\mathbf{x}$ by assigning it the label most frequently represented among the k nearest samples; in other words, a decision is made by examining the labels on the k nearest neighbors and taking a vote (Fig. 4.15). We shall not go into a thorough analysis of the k-nearest-neighbor rule. However, by considering the two-class case with k odd (to avoid ties), we can gain some additional insight into these procedures.

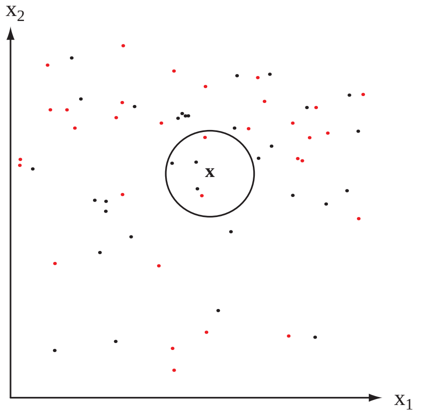

Figure 4.15: The k-nearest-neighbor query starts at the test point and grows a spherical region until it encloses k training samples, and labels the test point by a majority vote of these samples. In this $k = 5$ case, the test point $\mathbf{x}$ would be labelled the category of the black points.

The basic motivation for considering the k-nearest-neighbor rule rests on our earlier observation about matching probabilities with nature. We notice first that if k is fixed and the number n of samples is allowed to approach infinity, then all of the k nearest neighbors will converge to $\mathbf{x}$. Hence, as in the single-nearest-neighbor cases, the labels on each of the k-nearest-neighbors are random variables, which independently assume the values ω_i with probabilities $P(\omega_i|\mathbf{x}), i = 1, 2$. If $P(\omega_m|\mathbf{x})$ is the larger a posteriori probability, then the Bayes decision rule always selects ω_m. The single-nearest-neighbor rule selects ω_m with probability $P(\omega_m|\mathbf{x})$. The k-nearest-neighbor rule selects ω_m if a majority of the k nearest neighbors are labeled ω_m, an event of probability

$$\sum_{i=(k+1)/2}^{k} \binom{k}{i} P(\omega_m|\mathbf{x})^i [1 - P(\omega_m|\mathbf{x})]^{k-i}. \tag{54}$$

In general, the larger the value of k, the greater the probability that ω_m will be selected.

We could analyze the k-nearest-neighbor rule in much the same way that we analyzed the single-nearest-neighbor rule. However, since the arguments become more involved and supply little additional insight, we shall content ourselves with stating the results. It can be shown that if k is odd, the large-sample two-class error rate for the k-nearest-neighbor rule is bounded above by the function $C_k(P^*)$, where $C_k(P^*)$ is defined to be the smallest concave function of P^* greater than

$$\sum_{i=0}^{(k-1)/2} \binom{k}{i} \left[(P^*)^{i+1}(1-P^*)^{k-i} + (P^*)^{k-i}(1-P^*)^{i+1}\right]. \tag{55}$$

Here the summation over the first bracketed term represents the probability of error due to i points coming from the category having the minimum probability and $k-i > i$ points from the other category. The summation over the second term in the brackets is the probability that $k - i$ points are from the minimum-probability category and $i + 1 < k - i$ from the higher probability category. Both of these cases constitute errors under the k-nearest-neighbor decision rule, and thus we must add them to find the full probability of error (Problem 18).

Figure 4.16 shows the bounds on the k-nearest-neighbor error rates for several values of k. As k increases, the upper bounds get progressively closer to the lower bound — the Bayes rate. In the limit as k goes to infinity, the two bounds meet and the k-nearest-neighbor rule becomes optimal.

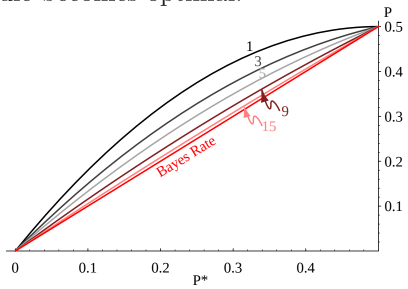

Figure 4.16: The error-rate for the k-nearest-neighbor rule for a two-category problem is bounded by $C_k(P^*)$ in Eq. 55. Each curve is labelled by k; when $k = \infty$, the estimated probabilities match the true probabilities and thus the error rate is equal to the Bayes rate, i.e., $P = P^*$.

At the risk of sounding repetitive, we conclude by commenting once again on the finite-sample situation encountered in practice. The k-nearest-neighbor rule can be viewed as another attempt to estimate the a posteriori probabilities $P(\omega_i|\mathbf{x})$ from samples. We want to use a large value of k to obtain a reliable estimate. On the other hand, we want all of the k nearest neighbors $\mathbf{x}'$ to be very near $\mathbf{x}$ to be sure that $P(\omega_i|\mathbf{x}')$ is approximately the same as $P(\omega_i|\mathbf{x})$. This forces us to choose a compromise k that is a small fraction of the number of samples. It is only in the limit as n goes to infinity that we can be assured of the nearly optimal behavior of the k-nearest-neighbor rule.

4.5.5 Computational Complexity of the k–Nearest-Neighbor Rule

The computational complexity of the nearest-neighbor algorithm — both in space (storage of prototypes) and time (search) — has received a great deal of analysis. There are a number of elegant theorems from computational geometry on the construction of Voronoi tesselations and nearest-neighbor searches in one- and two-dimensional spaces. However, because the greatest use of nearest-neighbor techniques is for problems with many features, we concentrate on the more general d-dimensional case.

Suppose we have n labelled training samples in d dimensions, and seek to find the closest to a test point $\mathbf{x}$ ($k = 1$). In the most naive approach we inspect each stored point in turn, calculate its Euclidean distance to $\mathbf{x}$, retaining the identity only of the current closest one. Each distance calculation is $O(d)$, and thus this search

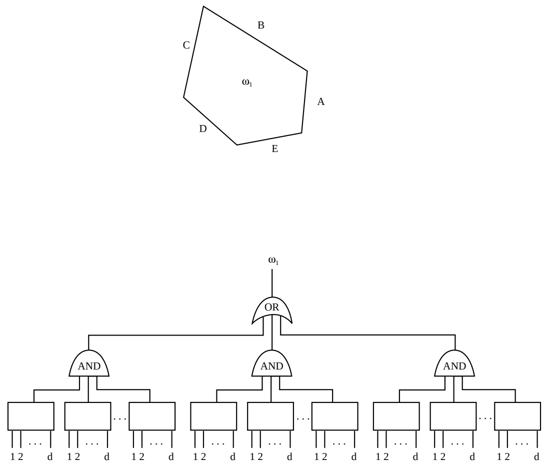

Figure 4.17: A parallel nearest-neighbor circuit can perform search in constant — i.e., $O(1)$ — time. The d-dimensional test pattern $\mathbf{x}$ is presented to each box, which calculates which side of a cell's face $\mathbf{x}$ lies on. If it is on the "close" side of every face of a cell, it lies in the Voronoi cell of the stored pattern, and receives its label.

is $O(dn^2)$. An alternative but straightforward parallel implementation is shown in Fig. 4.17, which is $O(1)$ in time and $O(n)$ in space.

There are three general algorithmic techniques for reducing the computational burden in nearest-neighbor searches: computing partial distances, prestructuring, and editing the stored prototypes. In *partial distance*, we calculate the distance using some subset r of the full d dimensions, and if this partial distance is too great we do not compute further. The partial distance based on r selected dimensions is

PARTIAL DISTANCE

$$D_r(\mathbf{a},\mathbf{b}) = \left(\sum_{k=1}^{r}(a_k - b_k)^2\right)^{1/2} \qquad (56)$$

where $r < d$. Intuitively speaking, partial distance methods assume that what we know about the distance in a subspace is indicative of the full space. Of course, the partial distance is strictly non-decreasing as we add the contributions from more and more dimensions. Consequently, we can confidently terminate a distance calculation to any prototype once its partial distance is greater than the full $r = d$ Euclidean distance to the current closest prototype.

In presturcturing we create some form of *search tree* in which prototypes are selectively linked. During classification, we compute the distance of the test point to one or a few stored "entry" or "root" prototypes and then consider only the prototypes linked to it. Of these, we find the one that is closest to the test point, and recursively consider only subsequent linked prototypes. If the tree is properly structured, we will reduce the total number of prototypes that need to be searched.

SEARCH TREE

Consider a trivial illustration of prestructuring in which we store a large number of prototypes that happen to be distributed uniformly in the unit square, i.e., $p(\mathbf{x}) \sim$

$U\left(\binom{0}{0},\binom{1}{1}\right)$. Imagine we prestructure this set using four entry or root prototypes — at $\binom{1/4}{1/4},\binom{1/4}{3/4},\binom{3/4}{1/4}$ and $\binom{3/4}{3/4}$ — each fully linked only to points in its corresponding quadrant. When a test pattern $\mathbf{x}$ is presented, the closest of these four prototypes is determined, and then the search is limited to the prototypes in the corresponding quadrant. In this way, 3/4 of the prototypes need never be queried.

Note that in this method we are no longer guaranteed to find the closest prototype. For instance, suppose the test point is near a boundary of the quadrants, e.g., $\mathbf{x} = \binom{0.499}{0.499}$. In this particular case only prototypes in the first quadrant will be searched. Note however that the closest prototype might actually be in one of the other three quadrants, somewhere near $\binom{0.5}{0.5}$. This illustrates a very general property in pattern recognition: the tradeoff of search complexity against accuracy.

More sophisticated search trees will have each stored prototype linked to a small number of others, and a full analysis of these methods would take us far afield. Nevertheless, here too, so long as we do not query all training prototypes, we are not guaranteed that the nearest prototype will be found.

EDITING

The third method for reducing the complexity of nearest-neighbor search is to eliminate "useless" prototypes during training, a technique known variously as *editing*, *pruning* or *condensing*. A simple method to reduce the $O(n)$ space complexity is to eliminate prototypes that are surrounded by training points of the same category label. This leaves the decision boundaries — and hence the error — unchanged, while reducing recall times. A simple editing algorithm is as follows.

Algorithm 3 (Nearest-neighbor editing)

```
begin initialize $j = 0, \mathcal{D} =$ data set, $n = \#$prototypes
        construct the full Voronoi diagram of $\mathcal{D}$
        do $j \leftarrow j + 1$;  for each prototype $\mathbf{x}'_j$
            Find the Voronoi neighbors of $\mathbf{x}'_j$
            if any neighbor is not from the same class as $\mathbf{x}'_j$ then mark $\mathbf{x}'_j$
        until $j = n$
  Discard all points that are not marked
  Construct the Voronoi diagram of the remaining (marked) prototypes
end
```

The complexity of this editing algorithm is $O(d^3 n^{\lfloor d/2 \rfloor} \ln n)$, where here the "floor" operation ($\lfloor\cdot\rfloor$) implies $\lfloor d/2 \rfloor = k$ if d is even, and $2k-1$ if d is odd (Problem 10).

According to Algorithm 3, if a prototype contributes to a decision boundary (i.e., at least one of its neighbors is from a different category), then it remains in the set; otherwise it is edited away (Problem 15). This algorithm does not guarantee that the *minimal* set of points is found (Problem 16), nevertheless, it is one of the examples in pattern recognition in which the computational complexity can be reduced — sometimes significantly — without affecting the accuracy. One drawback of such pruned nearest neighbor systems is that one generally cannot add training data later, since the pruning step requires knowledge of *all* the training data ahead of time (Computer exercise **??**). We conclude this section by noting the obvious, i.e., that we can combine these three complexity reduction methods. We might first edit the prototypes, then form a search tree during training, and finally compute partial distances during classification.

4.6 Metrics and Nearest-Neighbor Classification

The nearest-neighbor classifier relies on a metric or "distance" function between patterns. While so far we have assumed the Euclidean metric in d dimensions, the notion of a metric is far more general, and we now turn to the use alternate measures of distance to address key problems in classification. First let us review the properties of a metric. A metric $D(\cdot, \cdot)$ is merely a function that gives a generalized scalar distance between two argument patterns. A metric must have four properties: for all vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$

non-negativity: $D(\mathbf{a}, \mathbf{b}) \geq 0$

reflexivity: $D(\mathbf{a}, \mathbf{b}) = 0$ if and only if $\mathbf{a} = \mathbf{b}$

symmetry: $D(\mathbf{a}, \mathbf{b}) = D(\mathbf{b}, \mathbf{a})$

triangle inequality: $D(\mathbf{a}, \mathbf{b}) + D(\mathbf{b}, \mathbf{c}) \geq D(\mathbf{a}, \mathbf{c})$.

It is easy to verify that if the Euclidean formula for distance in d dimensions,

$$D(\mathbf{a}, \mathbf{b}) = \left(\sum_{k=1}^{d} (a_k - b_k)^2 \right)^{1/2}, \tag{57}$$

obeys the properties of metric. Moreover, if each coordinate is multiplied by an arbitrary constant, the resulting space also obeys a metric (Problem 19), though it can lead to problems in nearest-neighbor classifiers (Fig. 4.18).

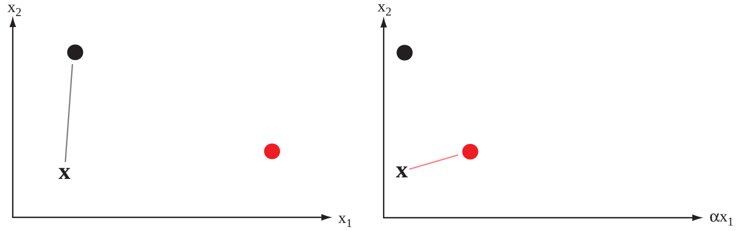

Figure 4.18: Even if each coordinate is scaled by some constant, the resulting space still obeys the properties of a metric. However, a nearest-neighbor classifier would have different results depending upon such rescaling. Consider the test point $\mathbf{x}$ and its nearest neighbor. In the original space (left), the black prototype is closest. In the figure at the right, the x_1 axis has been rescaled by a factor 1/3; now the nearest prototype is the red one. If there is a large disparity in the ranges of the full data in each dimension, a common procedure is to rescale all the data to equalize such ranges, and this is equivalent to changing the metric in the original space.

One general class of metrics for d-dimensional patterns is the Minkowski metric

Minkowsi metric

$$L_k(\mathbf{a}, \mathbf{b}) = \left(\sum_{i=1}^{d} |a_i - b_i|^k \right)^{1/k}, \tag{58}$$

also referred to as the L_k norm (Problem 20); thus, the Euclidean distance is the L_2 norm. The L_1 norm is sometimes called the *Manhattan* or *city block* distance, the shortest path between $\mathbf{a}$ and $\mathbf{b}$, each segment of which is parallel to a coordinate axis.

Manhattan distance

(The name derives from the fact that the streets of Manhattan run north-south and east-west.) Suppose we compute the distances between the projections of **a** and **b** onto each of the d coordinate axes. The L_∞ distance between **a** and **b** corresponds to the maximum of these projected distances (Fig. 4.19).

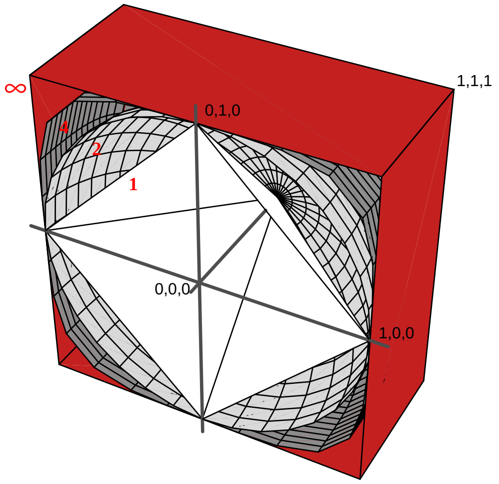

Figure 4.19: Each colored surface consists of points a distance 1.0 from the origin, measured using different values for k in the Minkowski metric (k is printed in red). Thus the white surfaces correspond to the L_1 norm (Manhattan distance), light gray the L_2 norm (Euclidean distance), dark gray the L_4 norm, and red the L_∞ norm.

Tanimoto metric

The *Tanimoto metric* finds most use in taxonomy, where the distance between two sets is defined as

$$D_{Tanimoto}(\mathcal{S}_1, \mathcal{S}_2) = \frac{n_1 + n_2 - 2n_{12}}{n_1 + n_2 - n_{12}}, \tag{59}$$

where n_1 and n_2 are the number of elements in sets $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively, and n_{12} is the number that is in both sets. The Tanimoto metric finds greatest use for problems in which two patterns or features — the elements in the set — are either the same or different, and there is no natural notion of graded similarity (Problem 27).

The selection among these or other metrics is generally dictated by computational concerns, and it is hard to base a choice on prior knowledge about the distributions. One exception is when there is great difference in the range of the data along different axes in a multidmensional data. Here, we should scale the data — or equivalently alter the metric — as suggested in Fig. 4.18.

4.6.1 Tangent distance

There may be drawbacks inherent in the uncritical use of a particular metric in nearest-neighbor classifiers, and these drawbacks can be overcome by the careful use of more general measures of distance. On crucial such problem is that of invariance. Consider a 100-dimensional pattern $\mathbf{x}'$ representing a 10×10 pixel grayscale image of a handwritten 5. Consider too the Euclidean distance from $\mathbf{x}'$ to the pattern representing an image that is shifted horizontally but otherwise identical (Fig. 4.20). Even if the relative shift is a mere three pixels, the Euclidean distance grows very large —

much greater than the distance to an unshifted 8. Clearly the Euclidean metric is of little use in a nearest-neighbor classifier that must be insensitive to such translations.

Likewise, other transformations, such as overall rotation or scale of the image, would not be well accommodated by Euclidean distance in this manner. Such drawbacks are especially pronounced if we demand that our classifier be simultaneously invariant to *several* transformations, such as horizontal translation, vertical translation, overall scale, rotation, line thickness, shear, and so on (Computer exercise 7 & 8). While we could preprocess the images by shifting their centers to coalign, then have the same bounding box, and so forth, such an approach has its own difficulties, such as sensitivity to outlying pixels or to noise. We explore here alternatives to such preprocessing.

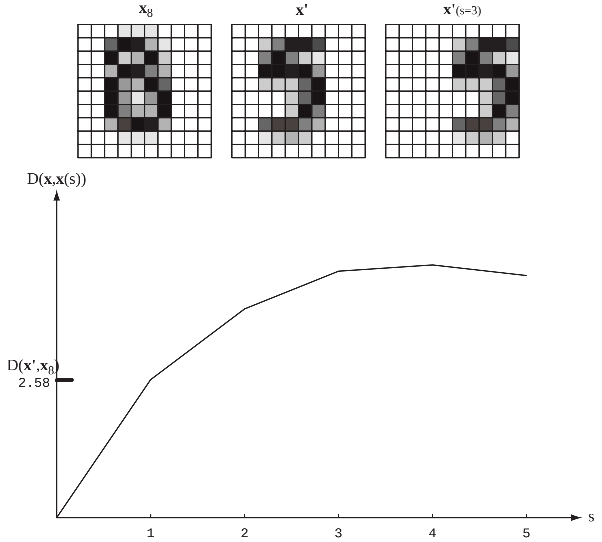

Figure 4.20: The uncritical use of Euclidean metric cannot address the problem of translation invariance. Pattern $\mathbf{x}'$ represents a handwritten 5, and $\mathbf{x}'(s = 3)$ the same shape but shifted three pixels to the right. The Euclidean distance $D(\mathbf{x}', \mathbf{x}'(s = 3))$ is much larger than $D(\mathbf{x}', \mathbf{x}_8)$, where $\mathbf{x}_8$ represents the handwritten 8. Nearest-neighbor classification based on the Euclidean distance in this way leads to very large errors. Instead, we seek a distance measure that would be insensitive to such translations, or indeed other known invariances, such as scale or rotation.

Ideally, during classification we would like to first transform the patterns to be as similar to one another and only then compute their similarity, for instance by the Euclidean distance. Alas, the computational complexity of such transformations make this ideal unattainable. Merely rotating a $k \times k$ image by a known amount and interpolating to a new grid is $O(k^2)$. But of course we do not know the proper rotation angle ahead of time and must search through several values, each value requiring a distance calculation to test the whether the optimal setting has been found. If we must search for the optimal set of parameters for *several* transformations for each stored prototype during classification, the computational burden is prohibitive (Problem 25).

The general approach in tangent distance classifiers is to use a novel measure of distance and a linear *approximation* to the arbitrary transforms. Suppose we believe there are r transformations applicable to our problem, such as horizontal translation, vertical translation, shear, rotation, scale, and line thinning. During construction of the classifier we take each stored prototype $\mathbf{x}'$ and perform each of the transformations $\mathcal{F}_i(\mathbf{x}';\ \alpha_i)$ on it. Thus $\mathcal{F}_i(\mathbf{x}';\ \alpha_i)$ could represent the image described by $\mathbf{x}'$, rotated by a small angle α_i. We then construct a *tangent vector* $\mathbf{TV}_i$ for each transformation:

TANGENT VECTOR

$$\mathbf{TV}_i = \mathcal{F}_i(\mathbf{x}';\ \alpha_i) - \mathbf{x}'. \tag{60}$$

While such a transformation may be compute intensive — as, for instance, the line thinning transform — it need be done only once, during training when computational constraints are lax. In this way we construct for each prototype $\mathbf{x}'$ an $r \times d$ matrix $\mathbf{T}$, consisting of the tangent vectors at $\mathbf{x}'$. (Such vectors can be orthonormalized, but we need assume here only that they are linearly independent.) It should be clear, too that this method will not work with binary images, since they lack a proper notion of derivative. If the data are binary, then, it is traditional to blur the images before creating a tangent distance based classifier.

Each point in the subspace spanned by the r tangent vectors passing through $\mathbf{x}'$ represents the linearized approximation to the full combination of transforms, as shown in Fig. 4.21. During classification we search for the point in the tangent space that is closest to a test point $\mathbf{x}$ — the linear approximation to our ideal. As we shall see, this search can be quite fast.

Now we turn to computing the tangent distance from a test point $\mathbf{x}$ to a particular stored prototype $\mathbf{x}'$. Formally, given a matrix $\mathbf{T}$ consisting of the r tangent vectors at $\mathbf{x}'$, the tangent distance from $\mathbf{x}'$ to $\mathbf{x}$ is

$$D_{tan}(\mathbf{x}', \mathbf{x}) = \min_{\mathbf{a}}[\|(\mathbf{x}' + \mathbf{Ta}) - \mathbf{x}\|], \tag{61}$$

i.e., the Euclidean distance from $\mathbf{x}$ to the tangent space of $\mathbf{x}'$. Equation 61 describes the so-called "one-sided" tangent distance, because only one pattern, $\mathbf{x}'$, is transformed. The two-sided tangent distance allows both $\mathbf{x}$ and $\mathbf{x}'$ to be transformed but improves the accuracy only slightly at a large added computational burden (Problem 23); for this reason we shall concentrate on the one-sided version.

During classification of $\mathbf{x}$ we will find its tangent distance to $\mathbf{x}'$ by finding the optimizing value of $\mathbf{a}$ required by Eq. 61. This minimization is actually quite simple, since the argument is a paraboloid as a function of $\mathbf{a}$, as shown in pink in Fig. 4.22. We find the optimal $\mathbf{a}$ via iterative gradient descent. For gradient descent we need the derivative of the (squared) Euclidean distance. The Euclidean distance in Eq. 61 obeys

$$D^2(\mathbf{x}' + \mathbf{Ta}, \mathbf{x}) = \|(\mathbf{x}' + \mathbf{Ta}) - \mathbf{x}\|^2, \tag{62}$$

and we compute the gradient with respect to the vector of parameters $\mathbf{a}$ — the projections onto the tangent vectors — as

$$\nabla_{\mathbf{a}} D^2(\mathbf{x}' + \mathbf{Ta}, \mathbf{x}) = 2\mathbf{T}^t(\mathbf{x}' + \mathbf{Ta} - \mathbf{x}). \tag{63}$$

Thus we can start with an arbitrary $\mathbf{a}$ and take a step in the direction of the negative gradient, updating our parameter vector as

$$\mathbf{a}(t+1) = \mathbf{a}(t) - \eta\mathbf{T}^t(\mathbf{Ta}(t) + \mathbf{x}' - \mathbf{x}), \tag{64}$$

where η is the scalar step size controlling the rate of convergence. So long as the step is not too large, we will reduce the squared Euclidean distance. When the minimum of such Euclidean distance is found, we have our tangent distance (Eq. 61). The

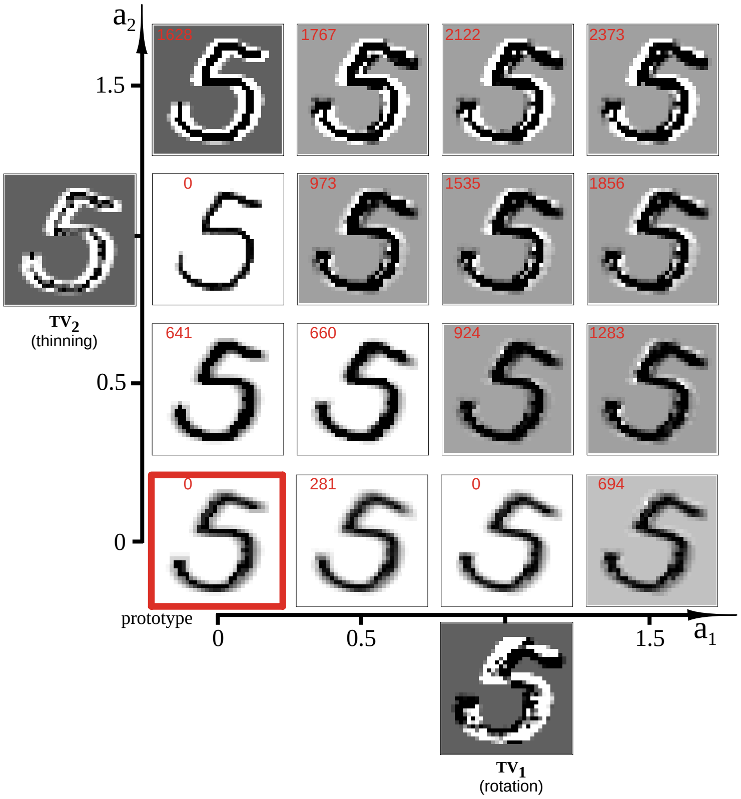

Figure 4.21: The pixel image of the handwritten 5 prototype at the lower left was subjected to two transformations, rotation, and line thinning, to obtain the tangent vectors $\mathbf{TV}_1$ and $\mathbf{TV}_2$; images corresponding to these tangent vectors are shown outside the axes. Each of the 16 images within the axes represents the prototype plus linear combination of the two tangent vectors with coefficients a_1 and a_2. The small red number in each image is the Euclidean distance between the tangent approximation and the image generated by the unapproximated transformations. Of course, this Euclidean distance is 0 for the prototype and for the cases $a_1 = 1, a_2 = 0$ and $a_1 = 0, a_2 = 1$. (The patterns generated with $a_1 + a_2 > 1$ have a gray background because of automatic grayscale conversion of images with negative pixel values.)

optimal **a** can also be found by standard matrix methods, but these generally have higher computational complexities, as is explored in Problems 21 & 22. We note that the methods for editing and prestructuring data sets described in Sec. 4.5.5 can be applied to tangent distance classifers too.

Nearest-neighbor classifiers using tangent distance have been shown to be highly accurate, but they require the designer to know which invariances and to be able to perform them on each prototype. Some of the insights from tangent approach can also be used for *learning* which invariances underly the training data — a topic we shall revisit in Chap. **??**.

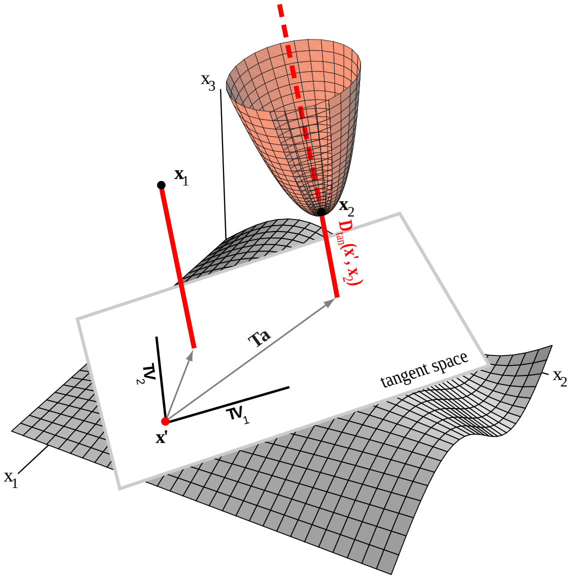

Figure 4.22: A stored prototype $\mathbf{x}'$, if transformed by combinations of two basic transformations, would fall somewhere on a complicated curved surface in the full d-dimensional space (gray). The tangent space at $\mathbf{x}'$ is an r-dimensional Euclidean space, spanned by the tangent vectors (here $\mathbf{TV}_1$ and $\mathbf{TV}_2$). The tangent distance $D_{tan}(\mathbf{x}', \mathbf{x})$ is the smallest Euclidean distance from $\mathbf{x}$ to the tangent space of $\mathbf{x}'$, shown in the solid red lines for two points, $\mathbf{x}_1$ and $\mathbf{x}_2$. Thus although the Euclidean distance from $\mathbf{x}'$ to $\mathbf{x}_1$ is less than to $\mathbf{x}_2$, for the tangent distance the situation is reversed. The Euclidean distance from $\mathbf{x}_2$ to the tangent space of $\mathbf{x}'$ is a quadratic function of the parameter vector $\mathbf{a}$, as shown by the pink paraboloid. Thus simple gradient descent methods can find the optimal vector $\mathbf{a}$ and hence the tangent distance $D_{tan}(\mathbf{x}', \mathbf{x}_2)$.

4.7 Fuzzy Classification

Occassionally we may have informal knowledge about a problem domain where we seek to build a classifier. For instance, we might feel, generally speaking, that an adult salmon is oblong and light in color, while a sea bass is stouter and dark. The approach taken in *fuzzy classification* is to create so-called "fuzzy category memberships functions," which convert an objectively measurable parameter into a subjective "category memberships," which are then used for classification. We must stress immediately that the term "categories" used by fuzzy practitioners refers not to the final class as we have been discussing, but instead just overlapping ranges of feature values. For instance, if we consider the feature value of lightness, fuzzy practitioners might split this into five "categories" — dark, medium-dark, medium, medium-light and light. In order to avoid misunderstandings, we shall use quotations when discussing such "categories."

For example we might have the lightness and shape of a fish be judged as in Fig. 4.23. Next we need a way to convert an objective measurement in several features into a category decision about the fish, and for this we need a *merging* or *conjunction rule* — a way to take the "category memberships" (e.g., lightness and shape) and yield a number to be used for making the final decision. Here fuzzy practitioners have

CONJUNCTION RULE

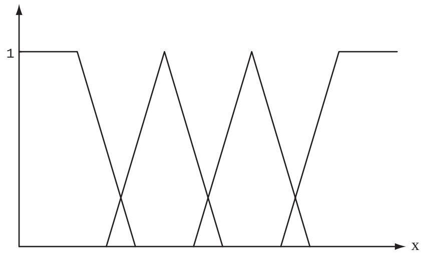

Figure 4.23: "Category membership" functions, derived from the designer's prior knowledge, together with a lead to discriminants. In this figure x might represent an objectively measureable value such as the reflectivity of a fish's skin. The designer believes there are four relevant ranges, which might be called dark, medium-dark, medium-light and light. Note, the memberships are not in true *categories* we wish to classify, but instead merely ranges of feature values.

at their disposal a large number of possible functions. Indeed, most functions can be used and there are few principled criteria to preference one over another. One guiding principle that is often invoked is that that in the extreme cases the membership functions have value 0 or 1, the conjunction reduces to standard predicate logic; likewise, symmetry in the arguments is virtually always assumed. Nevertheless, there are no strong principled reasons to impose these conditions, nor are they sufficient to determine the "categories."

Suppose the designer feels that the final category based on lightness ahd shape can be described as medium-light *and* oblong. While the heuristic category membership function ($\mu(\cdot)$) converts the objective measurements to two "category memberships," we now need a *conjunction rule* to transform the component "membership values" into a discriminant function. There are many ways to do this, but the most popular is

CONJUNCTION RULE

$$1 - Min[\mu_x(x), \mu_y(y)]. \tag{65}$$

and the obvious extension if there are more then two features.

It must be emphasized that fuzzy techniques are completely and thoroughly subsumed by the general notion of discriminant function discussed in Chap. **??** (Problem 29).

4.7.1 Are Fuzzy Category Memberships just Probabilities?

Even before the introduction of fuzzy methods and category membership functions, the statistics, pattern recognition and even mathematical philosophy communities argued a great deal over the fundamental nature of probability. Some questioned the applicability of the concept to single, non-repeatable events, feeling that statements about a single event — what was the probability of rain on Tuesday? — were meaningless. Such discussion made it quite clear that "probability" need not apply only to repeatable events. Instead, since the first half of the 20th century, probability has been used as the logic of reasonable inference — work that highlighted the notion of *subjective probability.* Moreover, pattern recognition practitioners had happily used

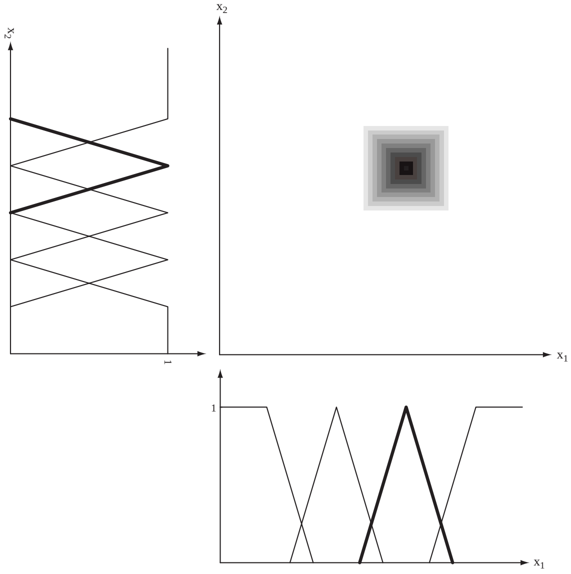

Figure 4.24: "Category membership" functions and a conjunction rule based on the designer's prior knowledge lead to discriminant functions. Here x_1 and x_2 are objectively measurable feature values. The designer believes that a particular class can be described as the conjunction of two "category memberships," here shown bold. Here the conjunction rule of Eq. 65 is used to give the discriminant function. The resulting discriminant function for the final category is indicated by the grayscale in the middle: the greater the discriminant, the darker. The designer constructs discriminant functions for other categories in a similar way (possibly also using disjunctions). During classification, the maximum discriminant function is chosen.

discriminant functions without concern over whether they represented probabilities, subjective probabilities, approximations to frequencies, or other fundamental entities.

While a full analysis of these topics would lead us away from our development of pattern recognition techniques, it pays to consider the claims of fuzzy logic proponents, since in order to be a good pattern recognition practitioner, we must understand what is or is not afforded by any technique. Proponents of fuzzy logic are adamant that category membership functions do not represent probabilities — subjective or not. Fuzzy practitioners point to examples such as when a half teaspoon of sugar is placed in a cup of tea, and conclude that the "membership" in the category *sweet* is 0.5, and that it would be incorrect to state that the *probability* the tea was sweet was 50%. But this situation be viewed simply as some sweetness feature value is 0.5, and there is some discriminant function, whose arguments include this feature value. One need not entertain xxx

Rather than debate the fundamental nature of probability, we should really be concerned with the nature of inference, i.e., how we take measurements and infer a category. Cox's axioms — sometimes called Cox$_{J}aynesaxioms---are$

1. If $P(a|d) > P(b|d)$ and $P(b|d) > P(c|d)$ then $P(a|d) > P(c|d)$. That is, degrees of belief have a natural ordering, given by real numbers.

2. $P(\text{not } a|d) = F_1[P(a|d)]$. That is, the degree of belief that a proposition is not the case is some function of the degree of belief that it *is* the case. Note that

such degrees of belief are graded values.

3. $P(a, b|d) = F_2[P(a|d), P(b|a, d)]$

The first axiom states merely that the probability of *not* having proposition b given a, is some function F_1 of the probability of b given a. The second, though not as evident, is

From these two, along with classical inference, we get the laws of probability. Any consistent inference method is formally equivalent to standard probabilistic inference.

In spite of the arguments on such foundational issues, many practitioners are happy to use fuzzy logic feeling that "whatever works" should be part of their repertoire. It is important, therefore, to understand the methodological strengths and limitations of the method. The limitations are formidable:

- Fuzzy methods are of very limited use in high dimensions or on complex problems. Pure fuzzy methods contribute little or nothing to problems with dozens or hundreds of features, and where there is training data.

- The amount of information the designer can be expected to bring to a problem is quite limited — the number, positions and widths of "category memberships."

- Because of their lack of normalization, pure fuzzy methods are poorly suited to problems in which there is a changing cost matrix λ_{ij} (Computer exercise 9).

- Pure fuzzy methods do not make use of training data. When such pure fuzzy methods (as outlined above) have unacceptable performance, it has been traditional to try to graft on adaptive (e.g., "neuro-fuzzy") methods.

If there is a contribution of fuzzy approaches to pattern recognition, it would lie in giving the steps by which one takes knowledge in a linguistic form and casts it into discriminant functions. It is unlikely that the verbal knowledge could extend to problems with dozens — much less hundreds — of features, the domain of the majority of real-world pattern recognition problems. A severe limitation of pure fuzzy methods is they do not rely on data, and when unsatisfactory results on problems of moderate size, it has been traditional to try to use neural or other adaptive techniques to compensate. At best, these are equivalent to maximum likelihood methods.

4.8 Relaxation methods

We have seen how the Parzen-window method uses a fixed window throughout the feature space, and that this could lead to difficulties: in some regions a small window width was appropriate while elsewhere a large one would be best. The k-nearest-neighbor method addressed this problem by adjusting the region based on the density of the points. Informally speaking, an approach that is intermediate between these two is to adjust the size of the window during training according to the distance to the nearest point of a *different* category. This is the method of some relaxation techniques. (The term "relaxation" refers to the underlying mathematical techniques for setting the parameters; we will consider only such relaxation issues, and concentrate instead on their effects.)

The simplest method is that of *potential functions* — which merely consists of an interpolation function. The difference with Parzen windows is that the magnitude

POTENTIAL FUNCTION

of each is adjusted so as to properly classify the training data. One representative method — called the *reduced coulomb energy* or RCE network — has the form shown in Fig. 4.25, which has the same topology as a Probabilistic neural network (Fig. 4.9).

REDUCED COULOMB ENERGY

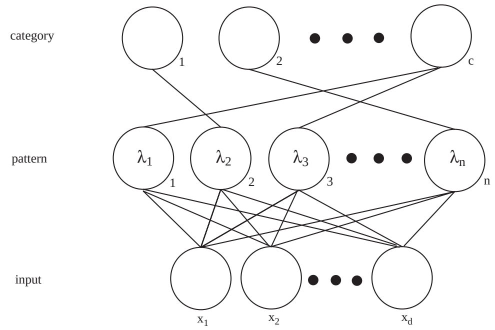

Figure 4.25: An RCE network is topologically equivalent to the PNN of Fig. 4.9. During training the wghts are adjusted to have the same values as the pattern presented, just as in a PNN. However, pattern units in an RCE network also have a modifiable "radius" parameter λ. During training, each λ is adjusted so that the region is as large as possible without containing training patterns from a different category.

The primary difference is that in an RCE network each pattern unit has an adjustable parameter that corresponds to the radius of the d-dimensional sphere. During training, each radius is adjusted so that each pattern unit covers a region as large as possible without containing a training point from another category.

Algorithm 4 (RCE training)

```
begin initialize $j = 0, n = \#\text{patterns}, \epsilon = \text{small param}, \lambda_m = \text{max radius}$
          do $j \leftarrow j + 1$
              train weight: $w_{jk} \leftarrow x_k$
              find nearest pt not in $\omega_i$: $\hat{\mathbf{x}} \leftarrow \arg \min_{\mathbf{x} \notin \omega_i} D(\mathbf{x}, \mathbf{x}')$
              set radius: $\lambda_j \leftarrow Min[D(\hat{\mathbf{x}}, \mathbf{x}') - \epsilon, \lambda_m]$
              if $\mathbf{x} \in \omega_i$ then $a_{ic} \leftarrow 1$
              until $j = n$
end
```

There are several subtleties that we need not consider right here. For instance, if the radius of a pattern unit becomes too small (i.e., less than some threshold λ_{min}), then it indicates that different categories are highly overlapping. In that case, the pattern unit is called a "probabilistic" unit, and so marked.

During classification, a test point is classified by the label of any point is by presenting the unit, getting activation. If probabilistic units overlap, Any region that is overlapped is considered ambiguous. Such ambiguous regions can be useful, since the teacher can be queried as to the identity of points in that region. If we continue to let λ_j be the radius around stored prototype $\mathbf{x}'_j$ and now let $\mathcal{D}_t$ be the set of stored prototypes in whose hypershperes test point $\mathbf{x}$ lies, then our classification algorithm is written as:

Algorithm 5 (RCE classification)

```
begin initialize $j = 0, k = 0, \mathbf{x} =$ test pattern, $\mathcal{D}_t = \{\}$
        do $j \leftarrow j + 1$
            if $D(\mathbf{x}, \mathbf{x}'_j) < \lambda_j$ then $\mathcal{D}_t \leftarrow \mathcal{D}_t \cup \mathbf{x}'_j$
            until $j = n$
            if cat of all $\mathbf{x}'_j \in \mathcal{D}_t$ is the same then return label of all $\mathbf{x}_k \in \mathcal{D}_t$
                                                                  else return "ambiguous" label
        end
```

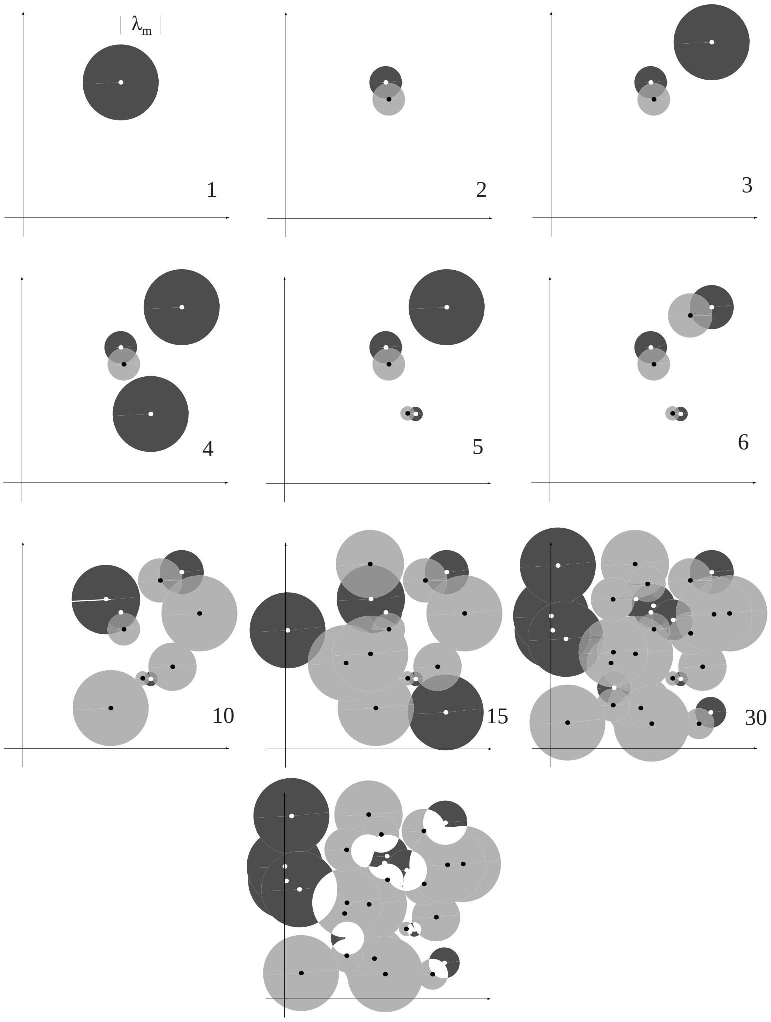

Figure 4.26: During training, each pattern has a parameter — equivalent to a radius in the d-dimensional space — that is adjusted to be as large as possible, without enclosing any points from a different category. As new patterns are presented, each such radius is decreased accordingly (and can never increase). In this way, each pattern unit can enclose several prototypes, but only those having the same category label. The number of points is shown in each component figure. The figure at the bottom shows the final complicated decision regions, colored by category.

4.9 Approximations by Series Expansions

The nonparametric methods described thus far suffer from the requirement that in general all of the samples must be stored or that the designer have extensive knowledge of the problem. Since a large number of samples is needed to obtain good estimates, the memory requirements can be severe. In addition, considerable computation time may be required each time one of the methods is used to estimate $p(\mathbf{x})$ or classify a new $\mathbf{x}$.

In certain circumstances the Parzen-window procedure can be modified to reduce these problems considerably. The basic idea is to approximate the window function by a finite series expansion that is acceptably accurate in the region of interest. If we are fortunate and can find two sets of functions $\psi_j(\mathbf{x})$ and $\chi_j(\mathbf{x})$ that allow the expansion

$$\varphi\Big(\frac{\mathbf{x}-\mathbf{x}_i}{h_n}\Big) = \sum_{j=1}^{m} a_j \psi_j(\mathbf{x})\chi_j(\mathbf{x}_i), \tag{66}$$

then we can split the dependence upon $\mathbf{x}$ and $\mathbf{x}_i$ as

$$\sum_{i=1}^{n}\varphi\Big(\frac{\mathbf{x}-\mathbf{x}_i}{h_n}\Big) = \sum_{j=1}^{m} a_j \psi_j(\mathbf{x}) \ \sum_{i=1}^{n}\chi_j(\mathbf{x}_i). \tag{67}$$

Then from Eq. 11 we have

$$p_n(\mathbf{x}) = \sum_{j=1}^{m} b_j \psi_j(\mathbf{x}), \tag{68}$$

where

$$b_j = \frac{a_j}{nV_n}\sum_{i=1}^{n}\chi_j(\mathbf{x}_i). \tag{69}$$

If a sufficiently accurate expansion can be obtained with a reasonable value for m, this approach has some obvious advantages. The information in the n samples is reduced to the m coefficients b_j. If additional samples are obtained, Eq. 69 for b_j can be updated easily, and the number of coefficients remains unchanged. If the functions $\psi_j(\cdot)$ and $\chi_j(\cdot)$ are polynomial functions of the components of $\mathbf{x}$ and $\mathbf{x}_i$, the expression for the estimate $p_n(\mathbf{x})$ is also a polynomial, which can be computed relatively efficiently. Furthermore, use of this estimate $p(\mathbf{x}|\omega_i)P(\omega_i)$ leads to a simple way of obtaining *polynomial discriminant functions*.

POLYNOMIAL DISCRIMINANT

Before becoming too enthusiastic, however, we should note one of the problems with this approach. A key property of a useful window function is its tendency to peak at the origin and fade away elsewhere. Thus $\varphi((\mathbf{x}-\mathbf{x}_i)/h_n)$ should peak sharply at $\mathbf{x}=\mathbf{x}_i$, and contribute little to the approximation of $p_n(\mathbf{x})$ for $\mathbf{x}$ far from $\mathbf{x}_i$. Unfortunately, polynomials have the annoying property of becoming unbounded. Thus, in a polynomial expansion we might find the terms associated with an $\mathbf{x}_i$ far from $\mathbf{x}$ contributing most (rather than least) to the expansion. It is quite important, therefore, to be sure that the expansion of each windowm function is in fact accurate in the region of interest, and this may well require a large number of terms.

There are many types of series expansions one might consider. Readers familiar with integral equations will naturally interpret Eq. 66 as an expansion of the kernel

EIGENFUNCTION

$\varphi(\mathbf{x}, \mathbf{x}_i)$ in a series of eigenfunctions. (In analogy with eigenvectors and eigenvalues, eigenfunctions are solutions to certain differential equations with fixed real-number coefficients.) Rather than computing eigenfunctions, one might choose any reasonable set of functions orthogonal over the region of interest and obtain a least-squares fit to the window function. We shall take an even more straightforward approach and expand the window function in a Taylor series. For simplicity, we confine our attention to a one-dimensional example using a Gaussian window function:

$$\begin{aligned} \sqrt{\pi}\ \varphi(u) &= e^{-u^2} \\ &\simeq \sum_{j=0}^{m-1} (-1)^j \frac{u^{2j}}{j!}. \end{aligned}$$

This expansion is most accurate near $u = 0$, and is in error by less than $u^{2m}/m!$. If we substitute $u = (x - x_i)/h$, we obtain a polynomial of degree $2(m-1)$ in x and x_i. For example, if $m = 2$ the window function can be approximated as

$$\begin{aligned} \sqrt{\pi}\varphi\Big(\frac{x - x_i}{h}\Big) &\simeq 1 - \Big(\frac{x - x_i}{h}\Big)^2 \\ &= 1 + \frac{2}{h^2} x\ x_i - \frac{1}{h^2}\ x^2 - \frac{1}{h^2}\ x_i^2, \end{aligned}$$

and thus

$$\sqrt{\pi} p_n(x) = \frac{1}{nh} \sum_{i=1}^{n} \sqrt{\pi}\varphi\Big(\frac{x - x_i}{h}\Big) \simeq b_0 + b_1 x + b_2 x^2, \qquad (70)$$

where the coefficients are

$$\begin{aligned} b_0 &= \frac{1}{h} - \frac{1}{h^3}\frac{1}{n}\sum_{i=1}^{n} x_i^2 \\ b_1 &= \frac{2}{h^3}\frac{1}{n}\sum_{i=1}^{n} x_i \\ b_2 &= -\frac{1}{h^3}. \end{aligned}$$

This simple expansion condenses the information in n samples into the values, b_0, b_1, and b_2. It is accurate if the largest value of $|x - x_i|$ is not greater than h. Unfortunately, this restricts us to a very wide window that is not capable of much resolution. By taking more terms we can use a narrower window. If we let r be the largest value of $|x - x_i|$ and use the fact that the error is the m-term expansion of $\sqrt{\pi}\ \varphi((x - x_i)/h)$ is less than $(r/h)^{2m} m!$, then using Stirling's approximation for $m!$ we find that the error in approximating $p_n(x)$ is less than

$$\frac{1}{\sqrt{\pi} h} \frac{\big(r/h\big)^{2m}}{m!} \simeq \frac{1}{\sqrt{\pi} h \sqrt{2\pi m}} \Big[\Big(\frac{e}{m}\Big) \Big(\frac{r}{h}\Big)^2 \Big]^m. \qquad (71)$$

Thus, the error becomes small only when $m > e(r/h)^2$. This implies the need for many terms if the window size h is small relative to the distance r from x to the most distant sample. Although this example is rudimentary, similar considerations arise in the multidimensional case even when more sophisticated expansions are used, and the procedure is most attractive when the window size is relatively large.

4.10 Fisher Linear Discriminant

One of the recurring problems encountered in applying statistical techniques to pattern recognition problems has been called the "curse of dimensionality." Procedures that are analytically or computationally manageable in low-dimensional spaces can become completely impractical in a space of 50 or 100 dimensions. Pure fuzzy methods are particularly ill-suited to such high-dimensional problems since it is implausible that the designer's linguistic intuition extends to such spaces. Thus, various techniques have been developed for reducing the dimensionality of the feature space in the hope of obtaining a more manageable problem.

We can reduce the dimensionality from d dimensions to one dimension if we merely project the d-dimensional data onto a line. Of course, even if the samples formed well-separated, compact clusters in d-space, projection onto an arbitrary line will usually produce a confused mixture of samples from all of the classes, and thus poor recognition performance. However, by moving the line around, we might be able to find an orientation for which the projected samples are well separated. This is exactly the goal of classical discriminant analysis.

Suppose that we have a set of n d-dimensional samples $\mathbf{x}_1, ..., \mathbf{x}_n$, n_1 in the subset $\mathcal{D}_1$ labelled ω_1 and n_2 in the subset $\mathcal{D}_2$ labelled ω_2. If we form a linear combination of the components of $\mathbf{x}$, we obtain the scalar dot product

$$y = \mathbf{w}^t\mathbf{x} \tag{72}$$

and a corresponding set of n samples $y_1, ..., y_n$ divided into the subsets $\mathcal{Y}_1$ and $\mathcal{Y}_2$. Geometrically, if $\|\mathbf{w}\| = 1$, each y_i is the projection of the corresponding $\mathbf{x}_i$ onto a line in the direction of $\mathbf{w}$. Actually, the magnitude of $\mathbf{w}$ is of no real significance, since it merely scales y. The direction of $\mathbf{w}$ is important, however. If we imagine that the samples labelled ω_1 fall more or less into one cluster while those labelled ω_2 fall in another, we want the projections falling onto the line to be well separated, not thoroughly intermingled. Figure 4.27 illustrates the effect of choosing two different values for $\mathbf{w}$ for a two-dimensional example. It should be abundantly clear that if the original distributions are multimodal and highly overlapping, even the "best" $\mathbf{w}$ is unlikely to provide adequate seaparation, and thus this method will be of little use.

We now turn to the matter of finding the best such direction $\mathbf{w}$, one we hope will enable accurate classification. A measure of the separation between the projected points is the difference of the sample means. If $\mathbf{m}_i$ is the d-dimensional sample mean given by

$$\mathbf{m}_i = \frac{1}{n_i} \sum_{\mathbf{x} \in \mathcal{D}_i} \mathbf{x}, \tag{73}$$

then the sample mean for the projected points is given by

$$\tilde{m}_i = \frac{1}{n_i} \sum_{y \in \mathcal{Y}_i} y$$

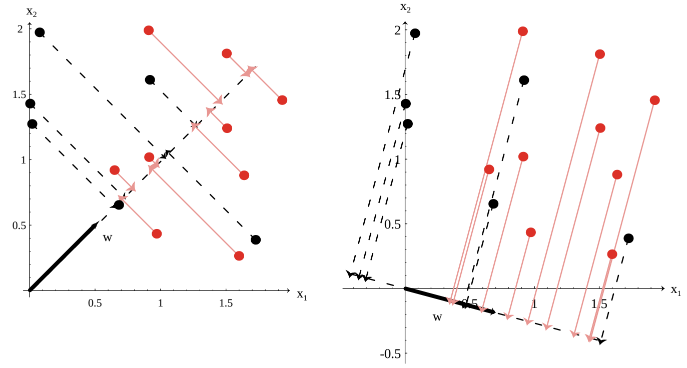

Figure 4.27: Projection of samples onto two different lines. The figure on the right shows greater separation between the red and black projected points.

$$= \frac{1}{n_i} \sum_{y \in \mathcal{Y}_i} \mathbf{w}^t \mathbf{x} = \mathbf{w}^t \mathbf{m}_i. \tag{74}$$

and is simply the projection of $\mathbf{m}_i$.

It follows that the distance between the projected means is

$$|\tilde{m}_1 \; - \; \tilde{m}_2| = |\mathbf{w}^t(\mathbf{m}_1 - \mathbf{m}_2)|, \tag{75}$$

and that we can make this difference as large as we wish merely by scaling $\mathbf{w}$. Of course, to obtain good separation of the projected data we really want the difference between the means to be large relative to some measure of the standard deviations for each class. Rather than forming sample variances, we define the *scatter* for projected samples labelled ω_i by

SCATTER

$$\tilde{s}_i^2 = \sum_{y \in \mathcal{Y}_i} (y - \tilde{m}_i)^2. \tag{76}$$

Thus, $(1/n)(\tilde{s}_1^2 + \tilde{s}_2^2)$ is an estimate of the variance of the pooled data, and $\tilde{s}_1^2 + \tilde{s}_2^2$ is called the total *within-class scatter* of the projected samples. The *Fisher linear discriminant* employs that linear function $\mathbf{w}^t\mathbf{x}$ for which the criterion function

WITHIN-CLASS SCATTER

$$J(\mathbf{w}) = \frac{|\tilde{m}_1 - \tilde{m}_2|^2}{\tilde{s}_1^2 + \tilde{s}_2^2} \tag{77}$$

is maximum (and independent of $\|\mathbf{w}\|$). While the $\mathbf{w}$ maximizing $J(\cdot)$ leads to the best separation between the two projected sets (in the sense just described), we will also need a threshold criterion before we have a true classifier. We first consider how to find the optimal $\mathbf{w}$, and later turn to the issue of thresholds.

To obtain $J(\cdot)$ as an explicit function of $\mathbf{w}$, we define the *scatter matrices* $\mathbf{S}_i$ and $\mathbf{S}_W$ by

SCATTER MATRICES

$$\mathbf{S}_i = \sum_{\mathbf{x} \in \mathcal{D}_i} (\mathbf{x} - \mathbf{m}_i)(\mathbf{x} - \mathbf{m}_i)^t \tag{78}$$

and

$$\mathbf{S}_W = \mathbf{S}_1 + \mathbf{S}_2. \tag{79}$$

Then we can write

$$\begin{aligned} \tilde{s}_i^2 &= \sum_{\mathbf{x}\in\mathcal{D}_i} (\mathbf{w}^t\mathbf{x} - \mathbf{w}^t\mathbf{m}_i)^2 \\ &= \sum_{\mathbf{x}\in\mathcal{D}_i} \mathbf{w}^t(\mathbf{x}-\mathbf{m}_i)(\mathbf{x}-\mathbf{m}_i)^t\mathbf{w} \\ &= \mathbf{w}^t\mathbf{S}_i\mathbf{w}; \end{aligned} \tag{80}$$

therefore the sum of these scatters can be written

$$\tilde{s}_1^2 + \tilde{s}_2^2 = \mathbf{w}^t\mathbf{S}_W\mathbf{w}. \tag{81}$$

Similarly, the separations of the projected means obeys

$$\begin{aligned} (\tilde{m}_1 - \tilde{m}_2)^2 &= (\mathbf{w}^t\mathbf{m}_1 - \mathbf{w}^t\mathbf{m}_2)^2 \\ &= \mathbf{w}^t(\mathbf{m}_1-\mathbf{m}_2)(\mathbf{m}_1-\mathbf{m}_2)^t\mathbf{w} \\ &= \mathbf{w}^t\mathbf{S}_B\mathbf{w}, \end{aligned} \tag{82}$$

where

$$\mathbf{S}_B = (\mathbf{m}_1-\mathbf{m}_2)(\mathbf{m}_1-\mathbf{m}_2)^t. \tag{83}$$

WITHIN-CLASS SCATTER

BETWEEN-CLASS SCATTER

We call $\mathbf{S}_W$ the *within-class scatter matrix.* It is proportional to the sample covariance matrix for the pooled d-dimensional data. It is symmetric and positive semidefinite, and is usually nonsingular if $n > d$. Likewise, $\mathbf{S}_B$ is called the *between-class scatter matrix.* It is also symmetric and positive semidefinite, but because it is the outer product of two vectors, its rank is at most one. In particular, for any $\mathbf{w}$, $\mathbf{S}_B\mathbf{w}$ is in the direction of $\mathbf{m}_1 - \mathbf{m}_2$, and $\mathbf{S}_B$ is quite singular.

In terms of $\mathbf{S}_B$ and $\mathbf{S}_W$, the criterion function $J(\cdot)$ can be written as

$$J(\mathbf{w}) = \frac{\mathbf{w}^t\mathbf{S}_B\mathbf{w}}{\mathbf{w}^t\mathbf{S}_W\mathbf{w}}. \tag{84}$$

This expression is well known in mathematical physics as the generalized Rayleigh quotient. It is easy to show that a vector $\mathbf{w}$ that maximizes $J(\cdot)$ must satisfy

$$\mathbf{S}_B\mathbf{w} = \lambda\mathbf{S}_W\mathbf{w}, \tag{85}$$

for some constant λ, which is a generalized eigenvalue problem (Problem 36). This can also be seen informally by noting that at an extremum of $J(\mathbf{w})$ a small change in $\mathbf{w}$ in Eq. 84 should leave unchanged the ratio of the numerator to the denominator. If $\mathbf{S}_W$ is nonsingular we can obtain a conventional eigenvalue problem by writing

$$\mathbf{S}_W^{-1}\mathbf{S}_B\mathbf{w} = \lambda\mathbf{w}. \tag{86}$$

In our particular case, it is unnecessary to solve for the eigenvalues and eigenvectors of $\mathbf{S}_W^{-1}\mathbf{S}_B$ due to the fact that $\mathbf{S}_B\mathbf{w}$ is always in the direction of $\mathbf{m}_1 - \mathbf{m}_2$. Since the scale factor for $\mathbf{w}$ is immaterial, we can immediately write the solution for the $\mathbf{w}$ that optimizes $J(\cdot)$:

$$\boxed{\mathbf{w} = \mathbf{S}_W^{-1}(\mathbf{m}_1 - \mathbf{m}_2).} \tag{87}$$

Thus, we have obtained $\mathbf{w}$ for Fisher's linear discriminant — the linear function yielding the maximum ratio of between-class scatter to within-class scatter. (The solution $\mathbf{w}$ given by Eq. 87 is sometimes called the *canonical variate.*) Thus the classification has been converted from a d-dimensional problem to a hopefully more manageable one-dimensional one. This mapping is many-to-one, and in theory can not possibly reduce the minimum achievable error rate if we have a very large training set. In general, one is willing to sacrifice some of the theoretically attainable performance for the advantages of working in one dimension. All that remains is to find the threshold, i.e., the point along the one-dimensional subspace separating the projected points.

When the conditional densities $p(\mathbf{x}|\omega_i)$ are multivariate normal with equal covariance matrices $\mathbf{\Sigma}$, we can calculate the threshold directly. In that case we recall (Chap. **??**, Sect. **??**) that the optimal decision boundary has the equation

$$\mathbf{w}^t\mathbf{x} + w_0 = 0 \tag{88}$$

where

$$\mathbf{w} = \mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2), \tag{89}$$

and where w_0 is a constant involving $\mathbf{w}$ and the prior probabilities. If we use sample means and the sample covariance matrix to estimate $\boldsymbol{\mu}_i$ and $\mathbf{\Sigma}$, we obtain a vector in the same direction as the $\mathbf{w}$ of Eq. 89 that maximized $J(\cdot)$. Thus, for the normal, equal-covariance case, the optimal decision rule is merely to decide ω_1 if Fisher's linear discriminant exceed some threshold, and to decide ω_2 otherwise. More generally, if we smooth the projected data, or fit it with a univariate Gaussian, we then should choose w_0 where the posteriors in the one dimensional distributions are equal.

The computational complexity of finding the optimal $\mathbf{w}$ for the Fisher linear discriminant (Eq. 87) is dominated by the calculation of the within-category total scatter and its inverse, an $O(d^2n)$ calculation.

4.11 Multiple Discriminant Analysis

For the c-class problem, the natural generalization of Fisher's linear discriminant involves $c-1$ discriminant functions. Thus, the projection is from a d-dimensional space to a $(c-1)$-dimensional space, and it is tacitly assumed that $d \geq c$. The generalization for the within-class scatter matrix is obvious:

$$\mathbf{S}_W = \sum_{i=1}^{c} \mathbf{S}_i \tag{90}$$

where, as before,

$$\mathbf{S}_i = \sum_{\mathbf{x}\in\mathcal{D}_i} (\mathbf{x} - \mathbf{m}_i)(\mathbf{x} - \mathbf{m}_i)^t \tag{91}$$

and

$$\mathbf{m}_i = \frac{1}{n_i}\sum_{\mathbf{x}\in\mathcal{D}_i}\mathbf{x}. \tag{92}$$

The proper generalization for $\mathbf{S}_B$ is not quite so obvious. Suppose that we define a *total mean vector* $\mathbf{m}$ and a *total scatter matrix* $\mathbf{S}_T$ by

TOTAL MEAN VECTOR

TOTAL SCATTER MATRIX

$$\mathbf{m} = \frac{1}{n}\sum_{\mathbf{x}}\mathbf{x} = \frac{1}{n}\sum_{i=1}^{c} n_i\mathbf{m}_i \tag{93}$$

and

$$\mathbf{S}_T = \sum_{\mathbf{x}}(\mathbf{x}-\mathbf{m})(\mathbf{x}-\mathbf{m})^t. \tag{94}$$

Then it follows that

$$\begin{aligned}\mathbf{S}_T &= \sum_{i=1}^{c}\sum_{\mathbf{x}\in\mathcal{D}_i}(\mathbf{x}-\mathbf{m}_i+\mathbf{m}_i-\mathbf{m})(\mathbf{x}-\mathbf{m}_i+\mathbf{m}_i-\mathbf{m})^t\\ &= \sum_{i=1}^{c}\sum_{\mathbf{x}\in\mathcal{D}_i}(\mathbf{x}-\mathbf{m}_i)(\mathbf{x}-\mathbf{m}_i)^t + \sum_{i=1}^{c}\sum_{\mathbf{x}\in\mathcal{D}_i}(\mathbf{m}_i-\mathbf{m})(\mathbf{m}_i-\mathbf{m})^t\\ &= \mathbf{S}_W + \sum_{i=1}^{c} n_i(\mathbf{m}_i-\mathbf{m})(\mathbf{m}_i-\mathbf{m})^t. \end{aligned} \tag{95}$$

It is natural to define this second term as a general between-class scatter matrix, so that the total scatter is the sum of the within-class scatter and the between-class scatter:

$$\mathbf{S}_B = \sum_{i=1}^{c} n_i(\mathbf{m}_i-\mathbf{m})(\mathbf{m}_i-\mathbf{m})^t \tag{96}$$

and

$$\mathbf{S}_T = \mathbf{S}_W + \mathbf{S}_B. \tag{97}$$

If we check the two-class case, we find that the resulting between-class scatter matrix is n_1n_2/n times our previous definition.*

The projection from a d-dimensional space to a $(c-1)$-dimensional space is accomplished by $c-1$ discriminant functions

$$y_i = \mathbf{w}_i^t\mathbf{x} \qquad i = 1, ..., c-1. \tag{98}$$

If the y_i are viewed as components of a vector $\mathbf{y}$ and the weight vectors $\mathbf{w}_i$ are viewed as the columns of a d-by-$(c-1)$ matrix $\mathbf{W}$, then the projection can be written as a single matrix equation

$$\mathbf{y} = \mathbf{W}^t\mathbf{x}. \tag{99}$$

* We could redefine $\mathbf{S}_B$ for the two-class case to obtain complete consistency, but there should be no misunderstanding of our usage.

The samples $\mathbf{x}_1, ..., \mathbf{x}_n$ project to a corresponding set of samples $\mathbf{y}_1, ..., \mathbf{y}_n$, which can be described by their own mean vectors and scatter matrices. Thus, if we define

$$\tilde{\mathbf{m}}_i = \frac{1}{n_i} \sum_{\mathbf{y} \in \mathcal{Y}_i} \mathbf{y} \tag{100}$$

$$\tilde{\mathbf{m}} = \frac{1}{n} \sum_{i=1}^{c} n_i \tilde{\mathbf{m}}_i \tag{101}$$

$$\tilde{\mathbf{S}}_W = \sum_{i=1}^{c} \sum_{\mathbf{y} \in \mathcal{Y}_i} (\mathbf{y} - \tilde{\mathbf{m}}_i)(\mathbf{y} - \tilde{\mathbf{m}}_i)^t \tag{102}$$

and

$$\tilde{\mathbf{S}}_B = \sum_{i=1}^{c} n_i (\tilde{\mathbf{m}}_i - \tilde{\mathbf{m}})(\tilde{\mathbf{m}}_i - \tilde{\mathbf{m}})^t, \tag{103}$$

it is a straightforward matter to show that

$$\tilde{\mathbf{S}}_W = \mathbf{W}^t \mathbf{S}_W \mathbf{W} \tag{104}$$

and

$$\tilde{\mathbf{S}}_B = \mathbf{W}^t \mathbf{S}_B \mathbf{W}. \tag{105}$$

These equations show how the within-class and between-class scatter matrices are transformed by the projection to the lower dimensional space (Fig. 4.28). What we seek is a transformation matrix $\mathbf{W}$ that in some sense maximizes the ratio of the between-class scatter to the within-class scatter. A simple scalar measure of scatter is the determinant of the scatter matrix. The determinant is the product of the eigenvalues, and hence is the product of the “variances” in the principal directions, thereby measuring the square of the hyperellipsoidal scattering volume. Using this measure, we obtain the criterion function

$$J(\mathbf{W}) = \frac{|\tilde{\mathbf{S}}_B|}{|\tilde{\mathbf{S}}_W|} = \frac{|\mathbf{W}^t \mathbf{S}_B \mathbf{W}|}{|\mathbf{W}^t \mathbf{S}_W \mathbf{W}|}. \tag{106}$$

The problem of finding a rectangular matrix $\mathbf{W}$ that maximizes $J(\cdot)$ is tricky, though fortunately it turns out that the solution is relatively simple. The columns of an optimal $\mathbf{W}$ are the generalized eigenvectors that correspond to the largest eigenvalues in

$$\mathbf{S}_B \mathbf{w}_i = \lambda_i \mathbf{S}_W \mathbf{w}_i. \tag{107}$$

A few observations about this solution are in order. First, if $\mathbf{S}_W$ is non-singular, this can be converted to a conventional eigenvalue problem as before. However, this is actually undesirable, since it requires an unnecessary computation of the inverse of $\mathbf{S}_W$. Instead, one can find the eigenvalues as the roots of the characteristic polynomial

$$|\mathbf{S}_B - \lambda_i \mathbf{S}_W| = 0 \tag{108}$$

and then solve

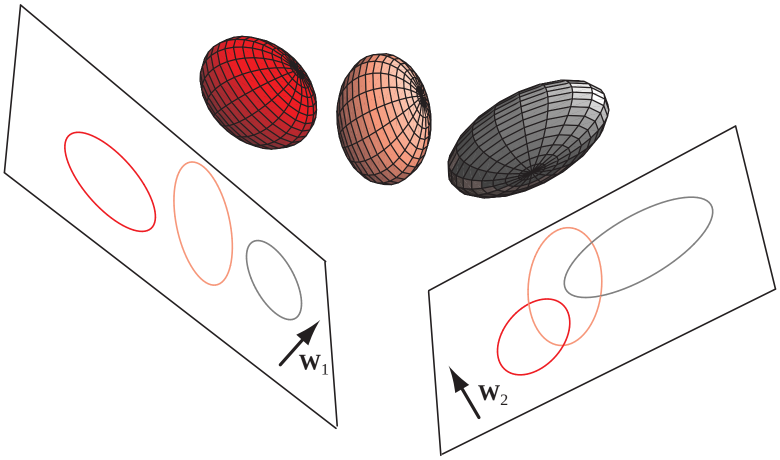

Figure 4.28: Three three-dimensional distributions are projected onto two-dimensional subspaces, described by a normal vectors $\mathbf{w}_1$ and $\mathbf{w}_2$. Informally, multiple discriminant methods seek the optimum such subspace, i.e., the one with the greatest separation of the projected distributions for a given total within-scatter matrix, here as associated with $\mathbf{w}_1$.

$$(\mathbf{S}_B - \lambda_i \mathbf{S}_W)\mathbf{w}_i = 0 \tag{109}$$

directly for the eigenvectors $\mathbf{w}_i$. Because $\mathbf{S}_B$ is the sum of c matrices of rank one or less, and because only $c-1$ of these are independent, $\mathbf{S}_B$ is of rank $c-1$ or less. Thus, no more than $c-1$ of the eigenvalues are nonzero, and the desired weight vectors correspond to these nonzero eigenvalues. If the within-class scatter is isotropic, the eigenvectors are merely the eigenvectors of $\mathbf{S}_B$, and the eigenvectors with nonzero eigenvalues span the space spanned by the vectors $\mathbf{m}_i - \mathbf{m}$. In this special case the columns of $\mathbf{W}$ can be found simply by applying the Gram-Schmidt orthonormalization procedure to the $c-1$ vectors $\mathbf{m}_i - \mathbf{m}$, $i = 1, ..., c-1$. Finally, we observe that in general the solution for $\mathbf{W}$ is not unique. The allowable transformations include rotating and scaling the axes in various ways. These are all linear transformations from a $(c-1)$-dimensional space to a $(c-1)$-dimensional space, however, and do not change things in any significant way; in particular, they leave the criterion function $J(\mathbf{W})$ invariant and the classifier unchanged.

If we have very little data, we would tend to project to a subspace of low dimension, while if there is more data, we can use a higher dimension, as we shall explore in Chap. **??**. Once we have projected the distributions onto the optimal subspace (defined as above), we can use the methods of Chapt. **??** to create our full classifier.

As in the two-class case, multiple discriminant analysis primarily provides a reasonable way of reducing the dimensionality of the problem. Parametric or nonparametric techniques that might not have been feasible in the original space may work well in the lower-dimensional space. In particular, it may be possible to estimate separate covariance matrices for each class and use the general multivariate normal assumption after the transformation where this could not be done with the original data. In general, if the transformation causes some unnecessary overlapping of the data and increases the theoretically achievable error rate, then the problem of classifying the data still remains. However, there are other ways to reduce the dimensionality of data, and we shall encounter this subject again in Chap. **??**. We note that there are also alternate methods of discriminant analysis — such as the selection of features based on statistical sigificance — some of which are given in the references for this chapter. Of these, Fisher's method remains a fundamental and widely used technique.

Summary

There are two overarching approaches to non-parametric estimation for pattern classification: in one the densities are estimated (and then used for classification), in the other the category is chosen directly. The former approach is exemplified by Parzen windows and their hardware implementation, Probabilistic neural networks. The latter is exemplified by k-nearest-neighbor and several forms of relaxation networks. In the limit of infinite training data, the nearest-neighbor error rate is bounded from above by twice the Bayes error rate. The extemely high space complexity of the nominal nearest-neighbor method can be reduced by editing (e.g., removing those prototypes that are surrounded by prototypes of the same category), prestructuring the data set for efficient search, or partial distance calculations. Novel distance measures, such as the tangent distance, can be used in the nearest-neighbor algorithm for incorporating known tranformation invariances.

Fuzzy classification methods employ heuristic choices of "category membership" and heuristic conjunction rules to obtain discriminant functions. Any benefit of such techniques is limited to cases where there is very little (or no) training data, small numbers of features, and when the knowledge can be gleaned from the designer's prior knowledge.

Relaxation methods such as potential functions create "basins of attraction" surrounding training prototypes; when a test pattern lies in such a basin, the corresponding prototype can be easily identified along with its category label. Reduced coloumb energy networks are one in the class of such relaxation networks, the basins are adjusted to be as large as possible yet not include prototypes from other categories.

The Fisher linear discriminant finds a good subspace in which categories are best separated; other techniques can then be applied in the subspace. Fisher's method can be extended to cases with multiple categories projected onto subspaces of higher dimension than a line.

Bibliographical and Historical Remarks

Parzen introduced his window method for estimating density functions [32], and its use in regression was pioneered by Ndaraya and Watson [**?**, **?**]. Its natural application to classification problems stems from the work of Specht [39], including its PNN hardware implementation [40].

Nearest-neighbor methods were first introduced by [16, 17], but it was over fifteen years later that computer power had increased, thereby making it practical and renewing interest in its theoretical foundations. Cover and Hart's foundational work on asymptotic bounds [10] were expanded somewhat through the analysis of Devroye [14]. The first pruning or editing work in [23] was followed by a number of related algorithms, such as that described in [5, 3]. The k-nearest neighbor was explored in [33]. The computational complexity of nearest neighbor (Voronoi) is described in [35]; work on search, as described in [27], has proven to be of greater use, in general. Much of the work on reducing the computational complexity of nearest-neighbor search comes from the vector quantization and compression community; for instance partial distance calculation are described in [21]. Friedman has an excellent analysis of some of the unintuitive properties of high dimensional spaces, and indirectly nearest neighbor classifiers, an inspiration for several problems here [19]. The definitive collection of seminal papers in nearest-neighbor classification is [12].

The notion of tangent distance was introduced by Simard and colleagues [38], and explored by a number of others [24]. Sperduti and Stork introduced a prestructuring and novel search criterion which speeds search in tangent based classifiers [41]. The greatest successes of tangent methods have been in optical character recognition, but the method can be applied in other domains, so long as the invariances are known. The study of general invariance has been most profitable when limited to a particular domain, and readers seeking further background should consult [31] for computer vision and [34] for speech. Background on image transformations is covered in [18].

The philosophical debate concerning frequency, probability, graded category membership, and so on, has a long history [29]. Keynes espoused a theory of probability as the logic of probable inference, and did not need to rely on the notion of repeatability, frequency, etc. We subscribe to the traditional view that probability is a conceptual and formal relation between hypotheses and conclusions — here, specifically between data and category. The limiting cases of such rational belief are certainty (on the one hand), and impossibility (on the other). Classical theory of probability cannot be based solely on classical logic, which has no formal notions for the probability of an event. While the rules in Keynes' probability [26] were taken as axiomatic, Cox [11] and later Jayne[**?**] sought to place a formal underpinning.

Many years after these debates, "fuzzy" methods were proposed from the computer science [43]. A formal equivalence of fuzzy category membership functions and probability is given in [22], which in turn is based on Cox [11]. Cheeseman has made some remarkably clear and forceful rebuttals to the assertions that fuzzy methods represent something beyond the notion of subjective probability [7, 8]; representative expositions to the contrary include [28, 4]. Readers unconcerned with foundational issues, and whether fuzzy methods provide any representational power or other benefits above standard probability (including subjective probability) can consult [25], which is loaded with over 3000 references. , many connectives for fuzzy logic [2]

Early reference on the use of potential functions for pattern classification is [1, 6]. This is closely allied with later work such as the RCE network described in [37, 36].

Fisher's early work on linear discriminants [15], is well described in [30] and a number of standard textbooks [9, 13, 20, 30, 42].

Problems

⊕ *Section 4.3*

1. Show that Eqs. 19–22 are sufficient to assure convergence in Eqs. 17 & 18.

2. Consider a normal $p(x) \sim N(\mu, \sigma^2)$ and Parzen-window function $\varphi(x) \sim N(0,1)$. Show that the Parzen-window estimate

$$p_n(x) = \frac{1}{nh_n}\sum_{i=1}^{n}\varphi\left(\frac{x-x_i}{h_n}\right),$$

has the following properties:

(a) $\bar{p}_n(x) \sim N(\mu, \sigma^2 + h_n^2)$

(b) $\text{Var}[p_n(x)] \simeq \frac{1}{2nh_n\sqrt{\pi}} p(x)$

(c) $p(x) - \bar{p}_n(x) \simeq \frac{1}{2}\left(\frac{h_n}{\sigma}\right)^2 \left[1 - \left(\frac{x-\mu}{\sigma}\right)^2\right] p(x)$

for small h_n. (Note: if $h_n = h_1/\sqrt{n}$, this shows that the error due to bias goes to zero as $1/n$, whereas the standard deviation of the noise only goes to zero as $\sqrt[4]{n}$.)

3. Let $p(x) \sim U(0, a)$ be uniform from 0 to a, and let a Parzen window be defined as $\varphi(x) = e^{-x}$ for $x > 0$ and 0 for $x \leq 0$.

(a) Show that the mean of such a Parzen-window estimate is given by

$$\bar{p}_n(x) = \begin{cases} 0 & x < 0 \\ \frac{1}{a}(1 - e^{-x/h_n}) & 0 \leq x \leq a \\ \frac{1}{a}(e^{a/h_n} - 1)e^{-x/h_n} & a \leq x. \end{cases}$$

(b) Plot $\bar{p}_n(x)$ versus x for $a = 1$ and $h_n = 1, 1/4$, and $1/16$.

(c) How small does h_n have to be to have less than one percent bias over 99 percent of the range $0 < x < a$?

(d) Find h_n for this condition if $a = 1$, and plot $\bar{p}_n(x)$ in the range $0 \leq x \leq 0.05$.

4. Suppose in a c-category supervised learning environment we sample the full distribution $p(\mathbf{x})$, and train a PNN classifier according to *Algorithm* **??**.

(a) Show that even if there are unequal category priors and hence unequal numbers of points in each category, the recognition method gives the right solution.

(b) Suppose we have trained a PNN with the assumption of equal category priors, but later wish to use it for a problem having the cost matrix λ_{ij}, representing the cost of choosing category ω_i when in fact the pattern came from ω_j. How should we do this?

(c) Suppose instead we know a cost matrix λ_{ij} *before* training. How shall we train a PNN for minimum risk?

⊕ *Section 4.4*

5. Show that Eq. 31 converges in probability to $p(\mathbf{x})$ given the conditions $\lim_{n\to\infty} k_n \to \infty$ and $\lim_{n\to\infty} k_n/n \to 0$.

6. Let $\mathcal{D} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ be a set of n independent labelled samples and let $\mathcal{D}_k(\mathbf{x}) = \{\mathbf{x}'_1, ..., \mathbf{x}'_k\}$ be the k nearest neighbors of $\mathbf{x}$. Recall that the k-nearest-neighbor rule for classifying $\mathbf{x}$ is to give $\mathbf{x}$ the label most frequently represented in $\mathcal{D}_k(\mathbf{x})$. Consider a two-category problem with $P(\omega_1) = P(\omega_2) = 1/2$. Assume further that the conditional densities $p(\mathbf{x}|\omega_i)$ are uniform within unit hyperspheres a distance of ten units apart.

(a) Show that if k is odd the average probability of error is given by

$$P_n(e) = \frac{1}{2^n} \sum_{j=0}^{(k-1)/2} \binom{n}{j}.$$

(b) Show that for this case the single-nearest neighbor rule has a lower error rate than the k-nearest-neighbor error rate for $k > 1$.

(c) If k is allowed to increases with n but is restricted by $k < a\sqrt{n}$, show that $P_n(e) \rightarrow 0$ as $n \rightarrow \infty$.

⊕ *Section 4.5*

7. Prove that the Voronoi cells induced by the single-nearest neighbor algorithm must always be convex. That is, for any two points $\mathbf{x}_1$ and $\mathbf{x}_2$ in a cell, all points on the line linking $\mathbf{x}_1$ and $\mathbf{x}_2$ must also lie in the cell.

8. It is easy to see that the nearest-neighbor error rate P can equal the Bayes rate P^* if $P^* = 0$ (the best possibility) or if $P^* = (c-1)/c$ (the worst possibility). One might ask whether or not there are problems for which $P = P^*$ when P^* is between these extremes.

(a) Show that the Bayes rate for the one-dimensional case where $P(\omega_i) = 1/c$ and

$$P(x|\omega_i) = \begin{cases} 1 & 0 \leq x \leq \frac{cr}{c-1} \\ 1 & i \leq x \leq i + 1 - \frac{cr}{c-1} \\ 0 & \text{elsewhere} \end{cases}$$

is $P^* = r$.

(b) Show that for this case that the nearest-neighbor rate is $P = P^*$.

9. Consider the following set of two-dimensional vectors:

ω_1		ω_2		ω_3	
x_1	x_2	x_1	x_2	x_1	x_2
10	0	5	10	2	8
0	-10	0	5	-5	2
5	-2	5	5	10	-4

(a) Plot the decision boundary resulting from the nearest-neighbor rule just for categorizing ω_1 and ω_2. Find the sample means $\mathbf{m}_1$ and $\mathbf{m}_2$ and on the same figure sketch the decision boundary corresponding to classifying $\mathbf{x}$ by assigning it to the category of the nearest sample mean.

(b) Repeat part (a) for categorizing only ω_1 and ω_3.

(c) Repeat part (a) for categorizing only ω_2 and ω_3.

(d) Repeat part (a) for a three-category classifier, classifying ω_1, ω_2 and ω_3.

10. Prove that the computational complexity of the basic nearest-neighbor editing algorith (*Algorithm* **??**) for n points in d dimension is $O(d^3 n^{\lfloor d/2 \rfloor} \ln n)$.

11. To understand the "curse of dimensionality" in greater depth, consider the effects of high dimensions on the simple nearest-neighbor algorithm. Suppose we need to estimate a density function $f(\mathbf{x})$ in the unit hypercube in $\mathbf{R}^d$ based on n samples. If $f(\mathbf{x})$ is complicated, we need dense samples to learn it well.

(a) Let n_1 denote the number of samples in a "dense" sample in $\mathbf{R}^1$. What is the sample size for the "same density" in $\mathbf{R}^d$? If $n_1 = 100$, what sample size is needed in a 20-dimensional space?

(b) Show that the interpoint distances are all large and roughly equal in $\mathbf{R}^d$, and that neighborhoods that have even just a few points must have large radii.

(c) Find $l_d(p)$, the length of a hypercube edge in d dimensions that contains the fraction p of points $(0 \leq p \leq 1)$. To better appreciate the implications of your result, calculate: $l_5(0.01)$, $l_5(0.1)$, $l_{20}(0.01)$, and $l_{20}(0.1)$.

(d) Show that nearly all points are close to an edge of the full space (e.g., the unit hypercube in d dimensions). Do this by calculating the L_∞ distance from one point to the closest other point. This shows that nearly all points are closer to an edge than to another training point. (Argue that L_∞ is more favorable than L_2 distance, even though it is easier to calculate here.) The result shows that most points are on or near the convex hull of training samples and that nearly every point is an "outlier" with respects to all the others.

12. Show how the "curse of dimensionality" (Problem 11) can be "overcome" by choosing or assuming that your model is of a particular sort. Suppose that we are estimating a function of the form $y = f(\mathbf{x}) + N(0, \sigma^2)$.

(a) Suppose the true function is linear, $f(\mathbf{x}) = \sum\limits_{j=1}^{n} a_j x_j$, and that the approximation is $\hat{f}(\mathbf{x}) = \sum\limits_{j=1}^{n} \hat{a}_j x_j$. Of course, the fit coefficients are:

$$\hat{a}_j = \arg\min_{a_j} \sum_{i=1}^{n} \left[y_i - \sum_{j=1}^{d} a_j x_{ij} \right]^2,$$

for $j = 1, \ldots, d$. Prove that $\mathcal{E}[f(\mathbf{x}) - \hat{f}(\mathbf{x})]^2 = d\sigma^2/n$, i.e., that it increases linearly with d, and not exponentially as the curse of dimensionality might otherwise suggest.

(b) Generalize your result from part (a) to the case where a function is expressed in a different basis set, i.e., $f(x) = \sum\limits_{i=1}^{n} a_i B_i(x)$ for some well-behaved basis set $B_i(x)$, and hence that the result does not depend on the fact that we have used a *linear* basis.

13. Consider classifiers based on samples from the distributions

$$\begin{aligned} p(x|\omega_1) &= \begin{cases} 2x & \text{for } 0 \leq x \leq 1 \\ 0 & \text{otherwise,} \end{cases} \\ \text{and} & \\ p(x|\omega_2) &= \begin{cases} 2 - 2x & \text{for } 0 \leq x \leq 1 \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

(a) What is the Bayes decision rule and the Bayes classification error?

(b) Suppose we randomly select a single point from ω_1 and a single point from ω_2, and create a nearest-neighbor classifier. Suppose too we select a test point from one of the categories (ω_1 for definiteness). Integrate to find the expected error rate $P_1(e)$.

(c) Repeat with two training samples from each category and a single test point in order to find $P_2(e)$.

(d) Generalize to find the arbitrary $P_n(e)$.

(e) Compare $\lim\limits_{n\to\infty} P_n(e)$ with the Bayes error.

14. Repeat Problem 13 but with

$$\begin{aligned} p(x|\omega_1) &= \begin{cases} 3/2 & \text{for } 0 \le x \le 2/3 \\ 0 & \text{otherwise,} \end{cases} \\ \text{and} & \\ p(x|\omega_2) &= \begin{cases} 3/2 & \text{for } 1/3 \le x \le 1 \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

15. Expand in greater detail Algorithm 3 and add a conditional branch that will speed it. Assuming the data points come from c categories and there are, on average, k Voronoi neighbors of any point $\mathbf{x}$, on average how much faster will your improved algorithm be?

16. Consider the simple nearest-neighbor editing algorithm (Algorithm 3).

(a) Show by counterexample that this algorithm does not yield the minimum set of points. (Hint: consider a problem where the points from each of two-categories are constrained to be on the intersections of a two-dimensional Cartesian grid.)

(b) Create a sequential editing algorithm, in which each point is considered in turn, and retained or rejected before the next point is considered. Prove that your algorithm does or does not depend upon the sequence the points are considered.

17. Consider classification problem where each of the c categories possesses the same distribution as well as prior $P(\omega_i) = 1/c$. Prove that the upper bound in Eq. 53, i.e.,

$$P \le P^* \left(2 - \frac{c}{c-1} P^* \right),$$

is achieved in this "zero-information" case.

18. Derive Eq. 55.

⊕ *Section 4.6*

19. Consider the Euclidean metric in d dimensions:

$$D(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{k=1}^{d} (a_k - b_k)^2}.$$

Suppose we rescale each axis by a fixed factor, i.e., let $x'_k = \alpha_k x_k$ for real, non-zero constants α_k, $k = 1, 2, ..., d$. Prove that the resulting space is a metric space. Discuss the import of this fact for standard nearest-neighbor classification methods.

20. Prove that the Minkowski metric indeed possesses the four properties required of all metrics.

21. Consider a non-iterative method for finding the tangent distance between $\mathbf{x}'$ and $\mathbf{x}$, given the matrix $\mathbf{T}$ consisting of the r (column) tangent vectors $\mathbf{TV}_i$ at $\mathbf{x}'$.

(a) As given in the text, take the gradient of the squared Euclidean distance in the **a** parameter space to find an equation that must be solved for the optimal **a**.

(b) Solve your first derivative equation to find the optimizing **a**.

(c) Compute the second derivative of $D^2(\cdot, \cdot)$ to prove that your solution must be a *minimum* squared distance, and not a maximum or inflection point.

(d) If there are r tangent vectors (invariances) in a d-dimensional space, what is the computational complexity of your method?

(e) In practice, the iterative method described in the text requires only a few (roughly 5) iterations for problems in handwritten OCR. Compare the complexities of your analytic solution to that of the iterative scheme.

22. Consider a tangent-distance based classifier based on n prototypes, each representing a $k \times k$ pixel pattern of a handwritten character. Suppose there are r invariances we believe characterize the problem. What is the storage requirements (space complexity) of such a tangent-based classifier?

23. The two-sided tangent distance allows *both* the stored prototype $\mathbf{x}'$ and the test point $\mathbf{x}$ to be transformed. Thus if $\mathbf{T}$ is the matrix of the r tangent vectors for $\mathbf{x}'$ and $\mathbf{S}$ likewise at $\mathbf{x}$, the two-sided tangent distance is

$$D_{2tan}(\mathbf{x}', \mathbf{x}) = \min_{\mathbf{a},\mathbf{b}} [\|(\mathbf{x}' + \mathbf{Ta}) - (\mathbf{x} + \mathbf{Sb})\|].$$

(a) Follow the logic in Problem 21 and calculate the gradient with respect to the **a** parameter vector and to the **b** parameter vector.

(b) What are the two update rules for an iterative scheme analogous to Eq. 64?

(c) Prove that there is a unique minium as a function of **a** and **b**. Describe this geometrically.

(d) In an iterative scheme, we would alternatively take steps in the **a** parameter space, then the **b** parameter space. What is the computational complexity to this approach to the two-sided tangent distance classifier?

(e) Why is the actual complexity for classification in a 2-sided tangent distance classifier even more sever than your result in (d) would suggest?

24. Consider the two-sided tangent distance described in Problem 23. Suppose we restrict ourselves to n prototypes $\mathbf{x}'$ in d dimensions, each with an associated matrix $\mathbf{T}$ of r tangent vectors, which we assume are linearly independent. Determine whether the two-sided tangent distance does or does not satisfy each of the requirements of a metric: non-negativity, reflexivity, symmetry and the triangle inequality.

25. Consider the computational complexity of nearest neighbor classifier for $k \times k$ pixel grayscale images of handwritten digits. Instead of using tangent distance, we will search for the parameters of full nonlinear transforms before computing a Euclidean distance. Suppose the number of operations needed to perform each of our r transformations (e.g., rotation, line thinning, shear, and so forth) is $a_i k^2$, where for the sake of simplicity we assume $a_i \simeq 10$. Suppose too that for the test of each prototype we must search though $A \simeq 5$ such values, and judge it by the Euclidean distance.

(a) Given a transformed image, how many operations are required to calculate the Euclidean distance to a stored prototype?

(b) Find the number of operations required per search.

(c) Suppose there are n prototypes. How many operations are required to find the nearest neighbor, given such transforms?

(d) Assume for simplicity that no complexity reduction methods have been used (such as editing, partial distance, graph creation). If the number of prototypes is $n = 10^6$ points, and there are $r = 6$ transformations, and basic operations on our computer require 10^{-9} seconds, how long does it take to classify a single point?

26. Explore the effect of r on the accuracy of nearest-neighbor search based on partial distance. Assume we have a large number n of points randomly placed in a d-dimensional hypercube. Suppose we have a test point $\mathbf{x}$, also selected randomly in the hypercume, and find its nearest neighbor. By definition, if we use the full d-dimensional Euclidean distance, we are guaranteed to find its nearest neighbor. Suppose though we use the partial distance

$$D_r(\mathbf{x}, \mathbf{x}') = \left(\sum_{i=1}^{r} (x_i - x_i')^2 \right)^{1/2}.$$

(a) Plot the probability that a partial distance search finds the true closest neighbor of an arbitrary point $\mathbf{x}$ as a function of r for fixed n ($1 \leq r \leq d$) for $d = 10$.

(b) Consider the effect of r on the accuracy of a nearest-neighbor classifier. Assume we have have $n/2$ prototypes from each two categories in a hypercube of length 1 on a side. The density for each category is separable into the product of (linear) ramp functions, highest at one side, and zero at the other side of the range. Thus the density for category ω_1 is highest at $(0, 0, ...0)^t$ and zero at $(1, 1, ..., 1)^t$, while the density for ω_2 is highest at $(1, 1, ..., 1)^t$ and zero at $(0, 0, ...0)^t$. State by inspection the Bayesian decision boundary.

(c) Calculate the Bayes error rate.

(d) Calculate the probability of correct classification of a point $\mathbf{x}$, randomly selected from one of the category densities, as a function of r in a partial distance metric.

(e) If $n = 10$, what must r be for the partial distance nearest neighbor classifier to be within 1% of the Bayes rate?

27. Consider the Tanimoto metric applied to sets having discrete elements.

(a) Determine whether the four properties of a metric are obeyed by $D_{Tanimoto}(\cdot, \cdot)$ as given in Eq. 59.

(b) Consider the following six words as mere sets of unordered letters: `pattern`, `pat`, `pots`, `stop`, `taxonomy` and `elementary`. Use the Tanimoto metric to rank order all $\binom{6}{2} = 30$ possible pairings of these sets.

(c) Is the triangle inequality obeyed for these six patterns?

⊕ *Section 4.7*

28. Suppose someone asks you whether a cup of water is hot or cold, and you respond that it is warm. Explain why this exchange in no way indicates that the membership of the cup in some "hot" class is a graded value less than 1.0.

29. Consider the design a fuzzy classifier for three types of fish based on two features: length and lightness. The designer feels that there are five ranges of length: short, medium-short, medium, medium-large and large. Similarly, lightness falls into three ranges: dark, medium and light. The designer uses the traingle function

$$\hat{T}(x;\ \mu_i, \delta_i) = \begin{cases} 1 - \frac{|x-\mu_i|}{\delta_i} & x \leq |\mu_i - \delta_i| \\ 0 & \text{otherwise.} \end{cases}$$

for the intermediate values, and an open triangle function for the extremes, i.e.,

$$\hat{C}(x, \mu_i, \delta_i) = \begin{cases} 1 & x > \mu_i \\ 1 - \frac{x-\mu_i}{\delta_i} & \mu_i - \delta_i \leq x \leq \mu_i \\ 0 & \text{otherwise,} \end{cases}$$

and its symmetric version.

Suppose we have for the length $\delta_i = 5$ and $\mu_1 = 5, \mu_2 = 7, \mu_3 = 9, \mu_4 = 11$ and $\mu_5 = 13$, and for lightness $\delta_j = 30$, $\mu_1 = 30, \mu_2 = 50$, and $\mu_3 = 70$. Suppose the designer feels that $\omega_1 =$ *medium-light and long*, $\omega_2 =$ *dark and short* and $\omega_3 =$ *medium dark and long*, where the conjunction rule "and" is defined in Eq. 65.

(a) Write the algebraic form of the discriminant functions.

(b) If every "category membership function" were rescaled by a constant, would classification change?

(c) Classify the pattern $\mathbf{x} = 7.5, 60$.

(d) Suppose that instead we knew that pattern is ω_x. Would we have any principled way to know whether the error was due to the number of category membership functions? their functional form? the conjunction rule?

⊕ *Section 4.8*

30. Suppose that through standard training of an RCE network (*Algorithm* 4), all the radii have been reduced to values less than λ_m. Prove that there is no subset of the training data that will yield the same category decision boundary.

⊕ *Section 4.9*

31. Consider a window function $\varphi(x) \sim N(0,1)$ and a density estimate

$$p_n(x) = \frac{1}{nh_n}\sum_{i=1}^{n} \varphi\Big(\frac{x - x_i}{h_n}\Big).$$

Approximate this estimate by factoring the window function and expanding the factor e^{x-x_i/h_n^2} in a Taylor series about the origin as follows:

(a) Show that in terms of the normalized variable $u = x/h_n$ the m-term approximation is given by

$$p_{nm}(x) = \frac{1}{\sqrt{2\pi}h_n} e^{-u^2/2} \sum_{j=0}^{m-1} b_j u^j$$

where

$$b_j = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{j!} u_i^j e^{-u_i^2/2}.$$

(b) Suppose that the n samples happen to be extremely tightly clustered about $u = u_0$. Show that the two-term approximation peaks at the two points where $u^2 + u/u_0 - 1 = 0$.

(c) Show that one peak occurs approximately at $u = u_0$, as desired, if $u_0 \ll 1$, but that it moves only to $u = 1$ for $u_0 \gg 1$.

(d) Confirm your answer to part (c) by plotting $p_{n2}(u)$ versus u for $u_0 = 0.01, 1,$ and 10. (Note: you may need to rescale the graphs vertically.)

⊕ *Section 4.10*

32. Let $p_x(\mathbf{x}|\omega_i)$ be arbitrary densities with means $\boldsymbol{\mu}_i$ and covariance matrices $\boldsymbol{\Sigma}_i$ — not necessarily normal — for $i = 1, 2$. Let $y = \mathbf{w}^t\mathbf{x}$ be a projection, and let the induced one-dimensional densities $p(y|\omega_i)$ have means μ_i and variances σ_i^2.

(a) Show that the criterion function

$$J_1(\mathbf{w}) = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}$$

is maximized by

$$\mathbf{w} = (\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2).$$

(b) If $P(\omega_i)$ is the prior probability for ω_i, show that

$$\begin{aligned} J_2(\mathbf{w}) &= \frac{(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^2}{P(\omega_1)\sigma_1^2 + P(\omega_2)\sigma_2^2} \quad \text{and} \\ \mathbf{w} &= [P(\omega_1)\boldsymbol{\Sigma}_1 + P(\omega_2)\boldsymbol{\Sigma}_2]^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2). \end{aligned}$$

(c) To which of these criterion functions is the $J(\mathbf{w})$ of Eq. **??** more closely related? Explain.

33. The expression

$$J_1 = \frac{1}{n_1 n_2} \sum_{y_i \in \mathcal{Y}_1} \sum_{y_j \in \mathcal{Y}_2} (y_i - y_j)^2$$

clearly measures the total within-group scatter.

(a) Show that this within-group scatter can be written as

$$J_1 = (m_1 - m_2)^2 + \frac{1}{n_1}s_1^2 + \frac{1}{n_2}s_2^2.$$

(b) Show that the total scatter is

$$J_2 = \frac{1}{n_1}s_1^2 + \frac{1}{n_2}s_2^2.$$

(c) If $y = \mathbf{w}^t\mathbf{x}$, show that the $\mathbf{w}$ optimizing J_1 subject to the constraint that $J_2 = 1$ is given by

$$\mathbf{w} = \lambda\Big(\frac{1}{n_1}\mathbf{S}_1 + \frac{1}{n_2}\mathbf{S}_2\Big)^{-1}(\mathbf{m}_1 - \mathbf{m}_2),$$

where

$$\begin{aligned} \lambda &= \Big[(\mathbf{m}_1 - \mathbf{m}_2)^t\Big(\frac{1}{n_1}\mathbf{S}_1 + \frac{1}{n_2}\mathbf{S}_2\Big)(\mathbf{m}_1 - \mathbf{m}_2)\Big]^{1/2}, \\ \mathbf{m}_i &= \frac{1}{n_i}\sum_{\mathbf{x}\in\mathcal{D}_i}\mathbf{x}, \end{aligned}$$

and

$$\mathbf{S}_i = \sum_{\mathbf{x}\in\mathcal{D}_i} n_i(\mathbf{m}_i - \mathbf{m})(\mathbf{m}_i - \mathbf{m})^t.$$

34. If $\mathbf{S}_B$ and $\mathbf{S}_W$ are two real, symmetric, d-by-d matrices, it is well known that there exists a set of n eigenvalues $\lambda_1, ..., \lambda_n$ satisfying $|\mathbf{S}_B - \lambda\mathbf{S}_W| = 0$, and a corresponding set of n eigenvectors $\mathbf{e}_1, ..., \mathbf{e}_n$ satisfying $\mathbf{S}_B\mathbf{e}_i = \lambda_i\mathbf{S}_W\mathbf{e}_i$. Furthermore, if $\mathbf{S}_W$ is positive definite, the eigenvectors can always be normalized so that $\mathbf{e}_i^t\mathbf{S}_W\mathbf{e}_j = \delta_{ij}$ and $\mathbf{e}_i^t\mathbf{S}_B\mathbf{e}_j = \lambda_i\delta_{ij}$. Let $\tilde{\mathbf{S}}_W = \mathbf{W}^t\mathbf{S}_W\mathbf{W}$ and $\tilde{\mathbf{S}}_B = \mathbf{W}^t\mathbf{S}_B\mathbf{W}$, where $\mathbf{W}$ is a d-by-n matrix whose columns correspond to n distinct eigenvectors.

(a) Show that $\tilde{\mathbf{S}}_W$ is the n-by-n identity matrix $\mathbf{I}$, and that $\tilde{\mathbf{S}}_B$ is a diagonal matrix whose elements are the corresponding eigenvalues. (This shows that the discriminant functions in multiple discriminant analysis are uncorrelated.)

(b) What is the value of $J = |\tilde{\mathbf{S}}_B|/|\tilde{\mathbf{S}}_W|$?

(c) Let $\mathbf{y} = \mathbf{W}^t\mathbf{x}$ be transformed by scaling the axes with a nonsingular n-by-n diagonal matrix $\mathbf{D}$ and by rotating this result with an orthogonal matrix $\mathbf{Q}$ where $\mathbf{y}' = \mathbf{QDy}$. Show that J is invariant to this transformation.

35. Consider two normal distributions with arbitrary but equal covariances. Prove that the Fisher linear discriminant, for suitable threshold, can be derived from the negative of the log-likelihood ratio.

36. Consider the criterion function $J(\mathbf{w})$ required for the Fisher linear discriminant.

(a) Fill in the steps leading from Eqs. 77, 79 & 83 to Eq. 84.

(b) Use matrix methods to show that the solution to Eq. 84 is indeed given by Eq. 85.

(c) At the extreme of $J(\mathbf{w})$, a small change in $\mathbf{w}$ must leave $J(\mathbf{w})$ unchanged. Consider a small perturbation away from the optimal, $\mathbf{w} + \Delta\mathbf{w}$, and derive the solution condition of Eq. 85.

⊕ *Section 4.11*

37. Consider multidiscriminant versions of Fisher's method for the case of c Gaussian distributions in d dimensions, each having the same covariance $\mathbf{\Sigma}$ (otherwise arbitrary) but different means. Solve for the optimal subspace in terms of $\mathbf{\Sigma}$ and the d mean vectors.

Computer exercises

Several exercises will make use of the following three-dimensional data sampled from three categories, denoted ω_i.

	ω_1			ω_2			ω_3		
sample	x_1	x_2	x_3	x_1	x_2	x_3	x_1	x_2	x_3
1	0.28	1.31	-6.2	0.011	1.03	-0.21	1.36	2.17	0.14
2	0.07	0.58	-0.78	1.27	1.28	0.08	1.41	1.45	-0.38
3	1.54	2.01	-1.63	0.13	3.12	0.16	1.22	0.99	0.69
4	-0.44	1.18	-4.32	-0.21	1.23	-0.11	2.46	2.19	1.31
5	-0.81	0.21	5.73	-2.18	1.39	-0.19	0.68	0.79	0.87
6	1.52	3.16	2.77	0.34	1.96	-0.16	2.51	3.22	1.35
7	2.20	2.42	-0.19	-1.38	0.94	0.45	0.60	2.44	0.92
8	0.91	1.94	6.21	-0.12	0.82	0.17	0.64	0.13	0.97
9	0.65	1.93	4.38	-1.44	2.31	0.14	0.85	0.58	0.99
10	-0.26	0.82	-0.96	0.26	1.94	0.08	0.66	0.51	0.88

⊕ *Section 4.2*

1. Explore some of the properties of density estimation in the following way.

(a) Write a program to generate points according to a uniform distribution in a unit cube, $-1/2 \leq x_i \leq 1/2$ for $i = 1, 2, 3$. Generate 10^4 such points.

(b) Write a program to estimate the density at the origin based on your 10^4 points as a function of the size of a cubical window function of size h. Plot your estimate as a function of h, for $0 < h \leq 1$.

(c) Evaluate the density at the origin using n of your points and the volume of a cube window which just encloses n points. Plot your estimate as a function of $n = 1, ..., 10^4$.

(d) Write a program to generate 10^4 points from a spherical Gaussian density (with $\mathbf{\Sigma} = \mathbf{I}$) centered on the origin. Repeat (b) & (c) with your Gaussian data.

(e) Discuss any qualitative differences between the functional dependencies of your estimation results for the uniform and Gaussian densities.

⊕ *Section 4.3*

2. Consider Parzen-window estimates and classifiers for points in the table above. Let your window function be a spherical Gaussian, i.e.,

$$\varphi((\mathbf{x} - \mathbf{x}_i)/h) \propto \text{Exp}[-(\mathbf{x} - \mathbf{x}_i)^t(\mathbf{x} - \mathbf{x}_i)/(2h^2)].$$

(a) Write a program to classify an arbitrary test point $\mathbf{x}$ based on the Parzen window estimates. Train your classifier using the three-dimensional data from your three categories in the table above. Set $h = 1$ and classify the following three points: $(0.50, 1.0, 0.0)^t$, $(0.31, 1.51, -0.50)^t$ and $(-0.3, 0.44, -0.1)^t$.

(b) Repeat with $h = 0.1$.

⊕ *Section 4.4*

3. Consider k-nearest-neighbor density estimations in different numbers of dimensions

(a) Write a program to find the k-nearest-neighbor density for n (unordered) points in one dimension. Use your program to plot such a density estimate for the x_1 values in category ω_3 in the table above for $k = 1, 3$ and 5.

(b) Write a program to find the k-nearest-neighbor density estimate for n points in two dimensions. Use your program to plot such a density estimate for the $x_1 - x_2$ values in ω_2 for $k = 1, 3$ and 5.

(c) Write a program to form a k-nearest-neighbor classifier for the three-dimensional data from the three categories in the table above. Use your program with $k = 1, 3$ and 5 to estimate the relative densities at the following points: $(-0.41, 0.82, 0.88)^t$, $(0.14, 0.72, 4.1)^t$ and $(-0.81, 0.61, -0.38)^t$.

⊕ *Section 4.5*

4. Write a program to create a Voronoi tesselation in two dimensions as follows.

(a) First derive analytically the equation of a line separating two arbitrary points.

(b) Given the full data set $\mathcal{D}$ of prototypes and a particular point $\mathbf{x} \in \mathcal{D}$, write a program to create a list of line segments comprising the Voronoi cell of $\mathbf{x}$.

(c) Use your program to form the Voronoi tesselation of the $x_1 - x_2$ features from the data of ω_1 and ω_3 in the table above. Plot your Voronoi diagram.

(d) Write a program to find the category decision boundary based on this full set $\mathcal{D}$.

(e) Implement a version of the pruning method described in *Algorithm* 3. Prune your data set from (c) to form a condensed set.

(f) Apply your programs from (c) & (d) to form the Voronoi tesselation and boundary for your condensed data set. Compare the decision boundaries you found for the full and the condensed sets.

5. Explore the tradeoff between computational complexity (as it relates to partial distance calculations) and search accuracy in nearest-neighbor classifiers in the following exercise.

(a) Write a program to generate n prototypes from a uniform distributions in a 6-dimensional hypercube centered on the origin. Use your program to generate 10^6 points for category ω_1, 10^6 different points for category ω_2, and likewise for ω_3 and ω_4. Denote this full set $\mathcal{D}$.

(b) Use your program to generate a test set $\mathcal{D}_t$ of $n = 100$ points, also uniformly distributed in the 6-dimensional hypercube.

(c) Write a program to implement the nearest-neighbor neighbor algorithm. Use this program to label each of your points in $\mathcal{D}_t$ by the category of its nearest neighbor in $\mathcal{D}$. From now on we will assume that the labels you find are in fact the true ones, and thus the "test error" is zero.

(d) Write a program to perform nearest-neighbor classification using partial distance, based on just the first r features of each vector. Suppose we define the search accuracy as the percentage of points in $\mathcal{D}_t$ that are associated with their particular closest prototype in $\mathcal{D}$. (Thus for $r = 6$, this accuracy is 100%, by construction.) For $1 \leq r \leq 6$ in your partial distance classifier, estimate the search accuracy. Plot a curve of this search accuracy versus r. What value of r would give a 90% search accuracy? (Round r to the nearest integer.)

(e) Estimate the "wall clock time" — the overall time required by your computer to perform the search — as a function of r. If T is the time for a full search in six dimensions, what value of r requires roughly $T/2$? What is the search accuracy in that case?

(f) Suppose instead we define search accuracy as the *classification* accuracy. Estimate this classification accuracy for a partial distance nearest-neighbor classifier using your points of $\mathcal{D}_t$. Plot this accuracy for $1 \leq r \leq 6$. Explain your result.

(g) Repeat (e) for this classification accuracy. If T is the time for full search in d dimensions, what value of r requires roughly $T/2$? What is the classification search accuracy in this case?

⊕ *Section 4.6*

6. Consider nearest-neighbor classifiers employing different values of k in the L_k norm or Minkowski metric.

(a) Write a program to implement a nearest-neighbor classifier for c categories, using the Minkowski metric or L_k norm, where k can be selected at classification time.

(b) Use the three dimensional data in the table above to classify the following points using the L_k norm for $k = 1, 2, 4$ and ∞: $(2.21, 1.9, 0.43)^t$, $(-0.15, 1.17, 6.19)^t$ and $(0.01, 1.34, 2.60)^t$.

7. Create a 10 × 10 pixel grayscale pattern $\mathbf{x}'$ of a handwritten 4.

(a) Plot the Euclidean distance between the 100-dimensional vectors corresponding to $\mathbf{x}'$ and a horizontally shifted version of it as a function of the horizontal offset.

(b) Shift $\mathbf{x}'$ by two pixels to the right to form the tangent vector $\mathbf{TV}_1$. Write a program to calculate the tangent distance for shifted patterns using your $\mathbf{TV}_1$. Plot the tangent distance as a function of the displacement of the test pattern. Compare your graphs and explain the implications.

8. Repeat Computer exercise 7 but for a handwritten 7, and vertical translations.

⊕ *Section 4.7*

9. Assume that size, color and shape are appropriate descriptions of fruit, and use fuzzy methods to classify fruit. In particular, assume all "category membership" functions are either triangular (with center μ and full half-width δ) or, at the extremes, are left- or right-open triangular functions.

Suppose the size features (measured in *cm*) are: small ($\mu = 2$), medium ($\mu = 4$), large ($\mu = 6$), and extra-large ($\mu = 8$). In all cases we assume the category membership tions have $\delta = 3$. Suppose shape is described by the excentricity, here the ratio of the major axis to minor axis lengths: thin ($\mu = 2, \delta = .6$), oblong ($\mu = 1.6, \delta = .3$), oval ($\mu = 1.4, \delta = .2$) and spherical ($\mu = 1.1, \delta = .2$). Suppose color here is represented by some measure of the mixture of red to yellow: yellow ($\mu = .1, \delta = .1$), yellow-orange ($\mu = 0.3, \delta = 0.3$), orange ($\mu = 0.5, \delta = 0.3$), orange-red ($\mu = 0.7, \delta = 0.3$) and red ($\mu = 0.9, \delta = 0.3$). The fuzzy practitioner believes the following are good descriptions of some common fruit:

- ω_1 = cherry = {small and spherical and red}
- ω_2 = orange = {medium and spherical and orange}
- ω_3 = banana = {large and thin and yellow}
- ω_4 = peach = {medium and spherical and orange-red}
- ω_5 = plum = {medium and spherical and red}
- ω_6 = lemon = {medium and oblong and yellow}
- ω_7 = grapefruit = {medium and spherical and yellow}

(a) Write a program to take any objective pattern and classify it.

(b) Classify each of these {size, shape, color}: $\{2.5, 1.0, 0.95\}$, $\{7.5, 1.9, 0.2\}$ and $\{5.0, 0.5, 0.4\}$.

(c) Suppose there is a cost associated with classification, as described by a cost matrix λ_{ij} — the cost of selecting ω_i given that the true category is ω_j. Suppose the cost matrix is

$$\lambda_{ij} = \begin{pmatrix} 0 & 1 & 1 & 0 & 2 & 2 & 1 \\ 1 & 0 & 2 & 2 & 0 & 0 & 1 \\ 1 & 2 & 0 & 1 & 0 & 0 & 2 \\ 0 & 2 & 1 & 0 & 2 & 2 & 2 \\ 2 & 0 & 0 & 2 & 0 & 1 & 1 \\ 2 & 0 & 0 & 2 & 1 & 0 & 2 \\ 1 & 1 & 2 & 2 & 1 & 2 & 0 \end{pmatrix}.$$

Reclassify the patterns in (b) for minimum cost.

⊕ *Section 4.8*

10. Explore relaxation networks in the following way.

(a) Write a program to implement an RCE classifier in three dimensions. Let the starting radius be $\lambda_m = 0.5$. Train your classifier with the data from the three categories in the table above. For this data, how many times was any sphere reduced in size? (If the same sphere is reduced two times, count that as twice.)

(b) Use your classifier to classify the following: $(0.53, -0.44, 1.1)^t$, $(-0.49, 0.44, 1.11)^t$ and $(0.51, -0.21, 2.15)^t$. If the classification of any point is ambiguous, state which are the candidate categories.

⊕ *Section 4.9*

11. Consider a classifier based on a Taylor series expansion of a Gaussian window function. Let k be the highest power of x_i in a Taylor series expansion of each of the independent features of a two-dimensional Gaussian. Below, consider just the $x_1 - x_2$ features of categories ω_2 and ω_3 in the table above.

(a) For each value $k = 2, 4,$ and 6, classify the following three points: $(0.56, 2.3, 0.10)^t$, $(0.60, 5.1, 0.86)^t$ and $(-0.95, 1.3, 0.16)^t$.

⊕ *Section 4.10*

12. Consider the Fisher linear discriminant method.

(a) Write a general program to calculate the optimal direction $\mathbf{w}$ for a Fisher linear discriminant based on three-dimensional data.

(b) Find the optimal $\mathbf{w}$ for categories ω_2 and ω_3 in the table above.

(c) Plot a line representing your optimal direction $\mathbf{w}$ and mark on it the positions of the projected points.

(d) In this subspace, fit each distribution with a (univariate) Gaussian, and find the resulting decision boundary.

(e) What is the training error (the error on the training points themselves) in the optimal subspace you found in (b)?

(f) For comparison, repeat (d) & (c) using instead the non-optimal direction $\mathbf{w} = (1.0, 2.0, -1.5)^t$. What is the training error in this non-optimal subspace?

⊕ *Section 4.11*

13. Consider the multicategory generalization of the Fisher linear discriminant, applied to the data in the table above.

(a) Write a general program to calculate the optimal $\mathbf{w}$ for multiple discriminant. Use your program to find the optimal two-dimensional plane (described by normal vector $\mathbf{w}$) for the three-dimensional data in the table.

(b) In the subspace, fit a circularly symmetric Gaussian to the data, and use a simple linear classifier in each to find the decision boundaries in the subspace.

(c) What is the error on the training set?

(d) Classify following points : $(1.40, -0.36, -0.41)^t$, $(0.62, 1.30, 1.11)^t$ and $(-0.11, 1.60, 1.51)^t$.

(e) For comparison, repeat (b) & (c) for the non-optimal direction $\mathbf{w} = (-0.5, -0.5, 1.0)^t$. Explain the difference between your training errors in the two cases.

Bibliography

[1] Mark A. Aizerman, E. M. Braverman, and L. I. Rozonoer. The Robbins-Monro process and the method of potential functions. *Automation and Remote Control*, 26:1882–1885, 1965.

[2] C. Alsina, E. Trillas, and L. Valverde. On some logical connectives for fuzzy set theory. *Journal of Mathematical Analysis and Applications*, 93(1):15–26, 1983.

[3] David Avis and Binay K. Bhattacharya. Algorithms for computing d-dimensional Voronoi diagrams and their duals. In Franco P. Preparata, editor, *Advances in Computing Research: Computational Geometry*, pages 159–180, Greenwich, CT, 1983. JAI Press.

[4] James C. Bezdek and Sankar K. Pal, editors. *Fuzzy Models for Pattern Recognition: Methods that search for structures in data.* IEEE Press, New York, NY, 1992.

[5] Binay K. Bhattacharya, Ronald S. Poulsen, and Godfried T. Toussaint. Application of proximity graphs to editing nearest neighbor decision rules. *Proc. 16th Symposium on the Interface Between Computer Science and Statistics*, pages 97–108, 1984.

[6] E. M. Braverman. On the potential function method. *Automation and Remote Control*, 26:2130–2138, 1965.

[7] Peter Cheeseman. In defense of probability. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1002–1009. Morgan Kaufmann, 1985.

[8] Peter Cheeseman. Probabilistic versus fuzzy reasoning. In Laveen N. Kanal and John F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, Amsterdam, The Netherlands, 1986. Elsevier Science Publishers.

[9] Herman Chernoff and Lincoln E. Moses. *Elementary Decision Theory.* John Wiley, New York, NY, 1959.

[10] Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, IT-13:21–27, 1967.

[11] Richard T. Cox. Probability, frequency, and reasonable expectation. *American Journal of Physics*, 14:1–13, 1946.

[12] Belur V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society, Washington, DC, 1991.

[13] Pierre A. Devijver and Josef Kittler. *Pattern Recognition: A Statistical Approach.* Prentice-Hall, London, UK, 1982.

[14] Luc P. Devroye. On the inequality of Cover and Hart in nearest neighbor discrimination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-3:75–78, 1981.

[15] Ronald A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7 Part II:179–188, 1936.

[16] Evelyn Fix and Joseph L. Hodges, Jr. Discriminatory analysis: Nonparametric discrimination: Consistency properties. *USAF School of Aviation Medicine*, 4:261–279, 1951.

[17] Evelyn Fix and Joseph L. Hodges, Jr. Discriminatory analysis: Nonparametric discrimination: Small sample performance. *USAF School of Aviation Medicine*, 11:280–322, 1952.

[18] James D. Foley, Andries Van Dam, Steven K. Feiner, and John F. Hughes. *Fundamentals of Interactive Computer Graphics: Principles and Practice.* Addison-Wesley, Reading, MA, 2 edition, 1990.

[19] Jerome H. Friedman. An overview of predictive learning and function approximation. In Vladimir Cherkassky, Jerome H. Friedman, and Harry Wechsler, editors, *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, New York, NY, 1994. Springer-Verlag, NATO ASI.

[20] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, New York, NY, 2nd edition, 1990.

[21] Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Processing.* Kluwer Academic Publishers, Boston, MA, 1992.

[22] Richard M. Golden. *Mathematical Methods for Neural Network Analysis and Design.* MIT Press, Cambridge, MA, 1996.

[23] Peter Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, IT-14(3):515–516, 1968.

[24] Trevor Hastie, Patrice Simard, and Eduard Säckinger. Learning prototype models for tangent distance. In Gerald Tesauro, David S. Touretzky, and Todd K. Leen, editors, *NIPS*, volume 7, pages 999–1006, Cambridge, MA, 1995. MIT Press.

[25] Abraham Kandel. *Fuzzy Techniques in Pattern Recognition.* Wiley, New York, NY, 1982.

[26] John Maynard Keynes. *A treatise on probability.* Macmillan, New York, NY, 1929.

[27] Donald E. Knuth. *The Art of Computer Programming*, volume 1. Addison-Wesley, Reading, MA, 1 edition, 1973.

[28] Bart Kosko. Fuzziness vs. probability. *International Journal of General Systems*, 17:211–240, 1990.

[29] Jan Lukasiewicz. Logical foundations of probability theory. In L. Berkowski, editor, *Jan Lukasiewicz: Selected works*. North-Holland, Amsterdam, 1970.

[30] Geoffrey J. McLachlan. *Discriminant Analysis and Statistical Pattern Recognition*. Wiley Interscience, New York, NY, 1992.

[31] Joseph L. Mundy and Andrews Zisserman, editors. *Geometric Invariance in Computer Vision*. MIT Press, Cambridge, MA, 1992.

[32] Emanuel Parzen. On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33:1065–1076, 1962.

[33] E. A. Patrick and F. P. Fischer, III. A generalized k-nearest neighbor rule. *Information and Control*, 16:128–152, 1970.

[34] Joseph S. Perkell and Dennis H. Klatt, editors. *Invariance and Variability in Speech Processes*. Lawrence Erlbaum, Hillsdale, NJ, 1986.

[35] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.

[36] Douglas L. Reilly and Leon N Cooper. An overview of neural networks: Early models to real world systems. In Steven F. Zornetzer, Joel L. Davis, Clifford Lau, and Thomas McKenna, editors, *An Introduction to Neural and Electronic Networks*, pages 229–250. Academic Press, New York, NY, 2nd edition, 1995.

[37] Douglas L. Reilly, Leon N Cooper, and C. Elbaum. A neural model for category learning. *Biological Cybernetics*, 45:35–41, 1982.

[38] Patrice Simard, Yann Le Cun, and John Denker. Efficient pattern recognition using a new transformation distance. In Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, editors, *NIPS*, volume 5, pages 50–58, San Mateo, CA, 1993. Morgan Kaufmann.

[39] Donald F. Specht. Generation of polynomial discriminant functions for pattern recognition. *IEEE Transactions on Electronic Computers*, EC-16:308–319, 1967.

[40] Donald F. Specht. Probabilistic neural networks. *Neural Networks*, 3:109–118, 1990.

[41] Alessandro Sperduti and David G. Stork. A rapid graph-based method for arbitrary transformation-invariant pattern classification. In Gerald Tesauro, David S. Touretzky, and Todd K. Leen, editors, *NIPS*, volume 7, pages 665–672, Cambridge, MA, 1995. MIT Press.

[42] Charles W. Therrien. *Decision Estimation and Classification: An Introduction to Pattern Recognition and Related Topics*. Wiley Interscience, New York, NY, 1989.

[43] Lotfi Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

Index