8 *Nonparametric Methods*

In the previous chapters, we discussed the parametric and semiparametric approaches where we assumed that the data is drawn from one or a mixture of probability distributions of known form. Now, we discuss the nonparametric approach that is used when no such assumption can be made about the input density and the data speaks for itself. We consider the nonparametric approaches for density estimation, classification, outlier detection, and regression and see how the time and space complexity can be checked.

8.1 Introduction

IN PARAMETRIC methods, whether for density estimation, classification, or regression, we assume a model valid over the whole input space. In regression, for example, when we assume a linear model, we assume that for any input, the output is the same linear function of the input. In classification when we assume a normal density, we assume that all examples of the class are drawn from this same density. The advantage of a parametric method is that it reduces the problem of estimating a probability density function, discriminant, or regression function to estimating the values of a small number of parameters. Its disadvantage is that this assumption does not always hold and we may incur a large error if it does not. If we cannot make such assumptions and cannot come up with a parametric model, one possibility is to use a semiparametric mixture model as we saw in chapter 7 where the density is written as a disjunction of a small number of parametric models.

NONPARAMETRIC ESTIMATION

In *nonparametric estimation*, all we assume is that *similar inputs have similar outputs*. This is a reasonable assumption: The world is smooth,

and functions, whether they are densities, discriminants, or regression functions, change slowly. Similar instances mean similar things. We all love our neighbors because they are so much like us.

Therefore, our algorithm is composed of finding the similar past instances from the training set using a suitable distance measure and interpolating from them to find the right output. Different nonparametric methods differ in the way they define similarity or interpolate from the similar training instances. In a parametric model, all of the training instances affect the final global estimate, whereas in the nonparametric case, there is no single global model; local models are estimated as they are needed, affected only by the nearby training instances.

Nonparametric methods do not assume any a priori parametric form for the underlying densities; in a looser interpretation, a nonparametric model is not fixed but its complexity depends on the size of the training set, or rather, the complexity of the problem inherent in the data.

INSTANCE-BASED
MEMORY-BASED
LEARNING

In machine learning literature, nonparametric methods are also called *instance-based* or *memory-based learning* algorithms, since what they do is store the training instances in a lookup table and interpolate from these. This implies that all of the training instances should be stored and storing all requires memory of $\mathcal{O}(N)$. Furthermore, given an input, similar ones should be found, and finding them requires computation of $\mathcal{O}(N)$. Such methods are also called *lazy* learning algorithms, because unlike the *eager* parametric models, they do not compute a model when they are given the training set but postpone the computation of the model until they are given a test instance. In the case of a parametric approach, the model is quite simple and has a small number of parameters, of order $\mathcal{O}(d)$, or $\mathcal{O}(d^2)$, and once these parameters are calculated from the training set, we keep the model and no longer need the training set to calculate the output. N is generally much larger than d (or d^2), and this increased need for memory and computation is the disadvantage of the nonparametric methods.

We start by estimating a density function, and discuss its use in classification. We then generalize the approach to regression.

8.2 Nonparametric Density Estimation

As usual in density estimation, we assume that the sample $\mathcal{X} = \{x^t\}_{t=1}^N$ is drawn independently from some unknown probability density $p(\cdot)$. $\hat{p}(\cdot)$

is our estimator of $p(\cdot)$. We start with the univariate case where x^t are scalars and later generalize to the multidimensional case.

The nonparametric estimator for the cumulative distribution function, $F(x)$, at point x is the proportion of sample points that are less than or equal to x

$$\hat{F}(x) = \frac{\#\{x^t \leq x\}}{N} \tag{8.1}$$

where $\#\{x^t \leq x\}$ denotes the number of training instances whose x^t is less than or equal to x. Similarly, the nonparametric estimate for the density function, which is the derivative of the cumulative distribution, can be calculated as

$$\hat{p}(x) = \frac{1}{h}\left[\frac{\#\{x^t \leq x+h\} - \#\{x^t \leq x\}}{N}\right] \tag{8.2}$$

h is the length of the interval and instances x^t that fall in this interval are assumed to be "close enough." The techniques given in this chapter are variants where different heuristics are used to determine the instances that are close and their effects on the estimate.

8.2.1 Histogram Estimator

HISTOGRAM

The oldest and most popular method is the *histogram* where the input space is divided into equal-sized intervals named *bins*. Given an origin x_o and a bin width h, the bins are the intervals $[x_o + mh, x_o + (m+1)h)$ for positive and negative integers m and the estimate is given as

$$\hat{p}(x) = \frac{\#\{x^t \text{ in the same bin as } x\}}{Nh} \tag{8.3}$$

In constructing the histogram, we have to choose both an origin and a bin width. The choice of origin affects the estimate near boundaries of bins, but it is mainly the bin width that has an effect on the estimate: With small bins, the estimate is spiky, and with larger bins, the estimate is smoother (see figure 8.1). The estimate is 0 if no instance falls in a bin and there are discontinuities at bin boundaries. Still, one advantage of the histogram is that once the bin estimates are calculated and stored, we do not need to retain the training set.

NAIVE ESTIMATOR

The *naive estimator* (Silverman 1986) frees us from setting an origin. It is defined as

$$\hat{p}(x) = \frac{\#\{x - h/2 < x^t \leq x + h/2\}}{Nh} \tag{8.4}$$

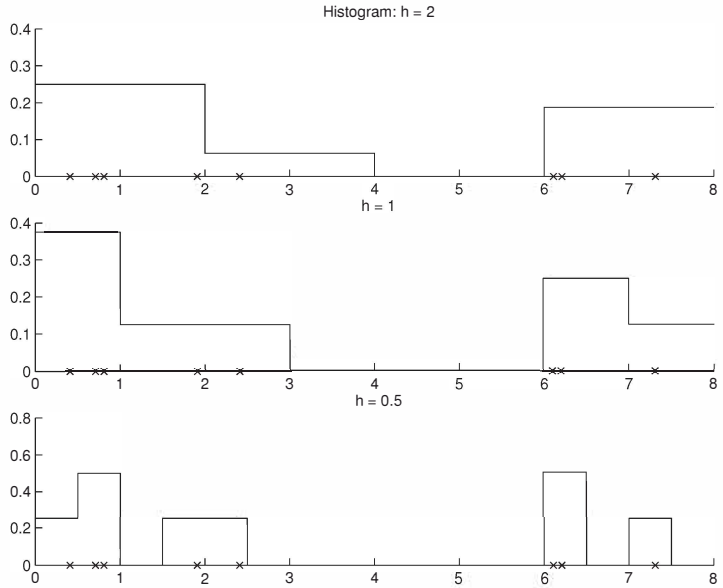

Figure 8.1 Histograms for various bin lengths. '×' denote data points.

and is equal to the histogram estimate where x is always at the center of a bin of size h (see figure 8.2). The estimator can also be written as

$$\hat{p}(x) = \frac{1}{Nh} \sum_{t=1}^{N} w\left(\frac{x - x^t}{h}\right) \tag{8.5}$$

with the *weight function* defined as

$$w(u) = \begin{cases} 1 & \text{if } |u| < 1/2 \\ 0 & \text{otherwise} \end{cases}$$

This is as if each x^t has a symmetric region of influence of size h around it and contributes 1 for an x falling in its region. Then the nonparametric estimate is just the sum of influences of x^t whose regions include x. Because this region of influence is "hard" (0 or 1), the estimate is not a continuous function and has jumps at $x^t \pm h/2$.

8.2.2 Kernel Estimator

KERNEL FUNCTION

To get a smooth estimate, we use a smooth weight function called a *kernel*

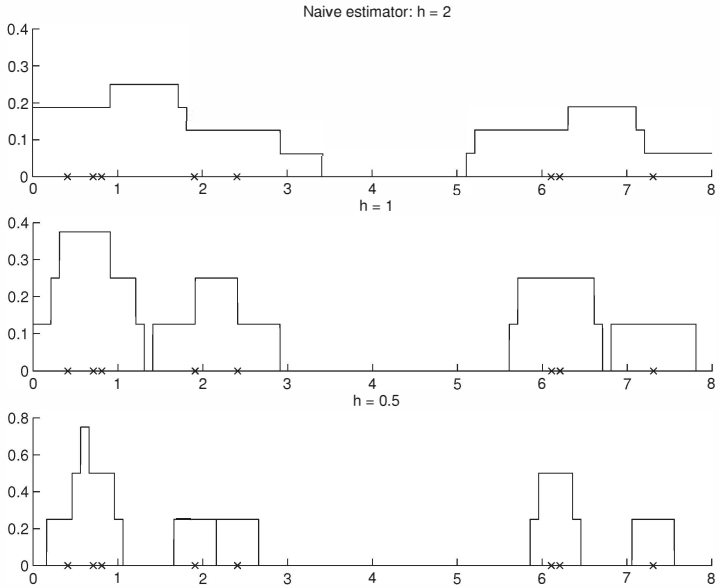

Figure 8.2 Naive estimate for various bin lengths.

function. The most popular is the Gaussian kernel:

$$K(u) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{u^2}{2}\right] \tag{8.6}$$

KERNEL ESTIMATOR
PARZEN WINDOWS

The *kernel estimator*, also called *Parzen windows*, is defined as

$$\hat{p}(x) = \frac{1}{Nh} \sum_{t=1}^{N} K\left(\frac{x - x^t}{h}\right) \tag{8.7}$$

The kernel function $K(\cdot)$ determines the shape of the influences and the window width h determines the width. Just like the naive estimate is the sum of "boxes," the kernel estimate is the sum of "bumps." All the x^t have an effect on the estimate at x, and this effect decreases smoothly as $|x - x^t|$ increases.

To simplify calculation, $K(\cdot)$ can be taken to be 0 if $|x - x^t| > 3h$. There exist other kernels easier to compute that can be used, as long as $K(u)$ is maximum for $u = 0$ and decreasing symmetrically as $|u|$ increases.

When h is small, each training instance has a large effect in a small region and no effect on distant points. When h is larger, there is more

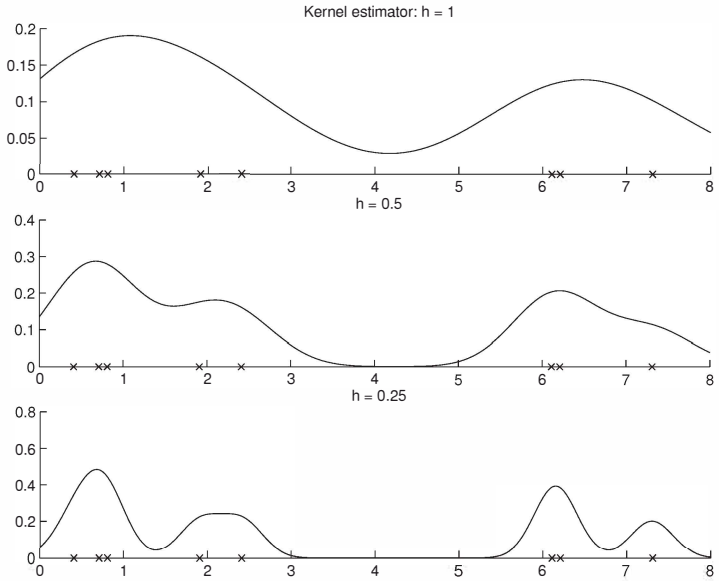

Figure 8.3 Kernel estimate for various bin lengths.

overlap of the kernels and we get a smoother estimate (see figure 8.3). If $K(\cdot)$ is everywhere nonnegative and integrates to 1, namely, if it is a legitimate density function, so will $\hat{p}(\cdot)$ be. Furthermore, $\hat{p}(\cdot)$ will inherit all the continuity and differentiability properties of the kernel $K(\cdot)$, so that, for example, if $K(\cdot)$ is Gaussian, then $\hat{p}(\cdot)$ will be smooth having all the derivatives.

One problem is that the window width is fixed across the entire input space. Various adaptive methods have been proposed to tailor h as a function of the density around x.

8.2.3 *k*-Nearest Neighbor Estimator

The nearest neighbor class of estimators adapts the amount of smoothing to the *local* density of data. The degree of smoothing is controlled by k, the number of neighbors taken into account, which is much smaller than N, the sample size. Let us define a distance between a and b, for example, $|a - b|$, and for each x, we define

$$d_1(x) \leq d_2(x) \leq \cdots \leq d_N(x)$$

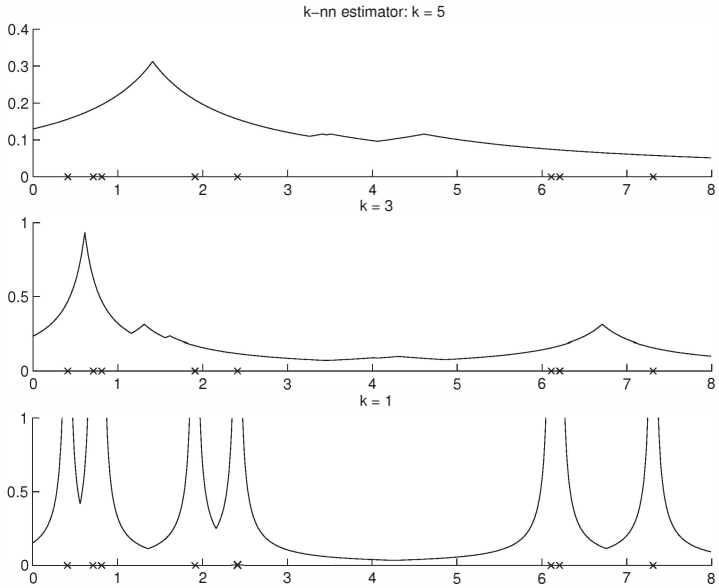

Figure 8.4 k-nearest neighbor estimate for various k values.

to be the distances arranged in ascending order, from x to the points in the sample: $d_1(x)$ is the distance to the nearest sample, $d_2(x)$ is the distance to the next nearest, and so on. If x^t are the data points, then we define $d_1(x) = \min_t |x - x^t|$, and if i is the index of the closest sample, namely, $i = \arg\min_t |x - x^t|$, then $d_2(x) = \min_{j \neq i} |x - x^j|$, and so forth.

k-NEAREST NEIGHBOR ESTIMATE

The *k-nearest neighbor* (k-nn) density estimate is

$$\hat{p}(x) = \frac{k}{2Nd_k(x)} \tag{8.8}$$

This is like a naive estimator with $h = 2d_k(x)$, the difference being that instead of fixing h and checking how many samples fall in the bin, we fix k, the number of observations to fall in the bin, and compute the bin size. Where density is high, bins are small, and where density is low, bins are larger (see figure 8.4).

The k-nn estimator is not continuous; its derivative has a discontinuity at all $\frac{1}{2}(x^{(j)} + x^{(j+k)})$ where $x^{(j)}$ are the order statistics of the sample. The k-nn is not a probability density function since it integrates to ∞, not 1.

To get a smoother estimate, we can use a kernel function whose effect decreases with increasing distance

$$\hat{p}(x) = \frac{1}{N d_k(x)} \sum_{t=1}^{N} K\left(\frac{x - x^t}{d_k(x)}\right) \tag{8.9}$$

This is like a kernel estimator with adaptive smoothing parameter $h = d_k(x)$. $K(\cdot)$ is typically taken to be the Gaussian kernel.

8.3 Generalization to Multivariate Data

Given a sample of d-dimensional observations $\mathcal{X} = \{\boldsymbol{x}^t\}_{t=1}^N$, the multivariate kernel density estimator is

$$\hat{p}(\boldsymbol{x}) = \frac{1}{N h^d} \sum_{t=1}^{N} K\left(\frac{\boldsymbol{x} - \boldsymbol{x}^t}{h}\right) \tag{8.10}$$

with the requirement that

$$\int_{\Re^d} K(\boldsymbol{x}) d\boldsymbol{x} = 1$$

The obvious candidate is the multivariate Gaussian kernel:

$$K(\boldsymbol{u}) = \left(\frac{1}{\sqrt{2\pi}}\right)^d \exp\left[-\frac{\|\boldsymbol{u}\|^2}{2}\right] \tag{8.11}$$

CURSE OF DIMENSIONALITY

However, care should be applied to using nonparametric estimates in high-dimensional spaces because of the *curse of dimensionality*: Let us say $\boldsymbol{x}$ is eight-dimensional, and we use a histogram with ten bins per dimension, then there are 10^8 bins, and unless we have lots of data, most of these bins will be empty and the estimates in there will be 0. In high dimensions, the concept of "close" also becomes blurry so we should be careful in choosing h.

For example, the use of the Euclidean norm in equation 8.11 implies that the kernel is scaled equally on all dimensions. If the inputs are on different scales, they should be normalized to have the same variance. Still, this does not take correlations into account, and better results are achieved when the kernel has the same form as the underlying distribution

$$K(\boldsymbol{u}) = \frac{1}{(2\pi)^{d/2}|\mathbf{S}|^{1/2}} \exp\left[-\frac{1}{2}\boldsymbol{u}^T \mathbf{S}^{-1} \boldsymbol{u}\right] \tag{8.12}$$

where $\mathbf{S}$ is the sample covariance matrix. This corresponds to using Mahalanobis distance instead of the Euclidean distance.

8.4 Nonparametric Classification

When used for classification, we use the nonparametric approach to estimate the class-conditional densities, $p(\boldsymbol{x}|C_i)$. The kernel estimator of the class-conditional density is given as

$$\hat{p}(\boldsymbol{x}|C_i) = \frac{1}{N_i h^d} \sum_{t=1}^{N} K\left(\frac{\boldsymbol{x}-\boldsymbol{x}^t}{h}\right) r_i^t \tag{8.13}$$

where r_i^t is 1 if $\boldsymbol{x}^t \in C_i$ and 0 otherwise. N_i is the number of labeled instances belonging to C_i: $N_i = \sum_t r_i^t$. The MLE of the prior density is $\hat{P}(C_i) = N_i/N$. Then, the discriminant can be written as

$$\begin{aligned} g_i(\boldsymbol{x}) &= \hat{p}(\boldsymbol{x}|C_i)\hat{P}(C_i) \\ &= \frac{1}{Nh^d} \sum_{t=1}^{N} K\left(\frac{\boldsymbol{x}-\boldsymbol{x}^t}{h}\right) r_i^t \end{aligned} \tag{8.14}$$

and $\boldsymbol{x}$ is assigned to the class for which the discriminant takes its maximum. The common factor $1/(Nh^d)$ can be ignored. So each training instance votes for its class and has no effect on other classes; the weight of vote is given by the kernel function $K(\cdot)$, typically giving more weight to closer instances.

For the special case of k-nn estimator, we have

$$\hat{p}(\boldsymbol{x}|C_i) = \frac{k_i}{N_i V^k(\boldsymbol{x})} \tag{8.15}$$

where k_i is the number of neighbors out of the k nearest that belong to C_i and $V^k(\boldsymbol{x})$ is the volume of the d-dimensional hypersphere centered at $\boldsymbol{x}$, with radius $r = \|\boldsymbol{x} - \boldsymbol{x}_{(k)}\|$ where $\boldsymbol{x}_{(k)}$ is the k-th nearest observation to $\boldsymbol{x}$ (among all neighbors from all classes of $\boldsymbol{x}$): $V^k = r^d c_d$ with c_d as the volume of the unit sphere in d dimensions, for example, $c_1 = 2, c_2 = \pi, c_3 = 4\pi/3$, and so forth. Then

$$\hat{P}(C_i|\boldsymbol{x}) = \frac{\hat{p}(\boldsymbol{x}|C_i)\hat{P}(C_i)}{\hat{p}(\boldsymbol{x})} = \frac{k_i}{k} \tag{8.16}$$

k-NN CLASSIFIER

The *k-nn classifier* assigns the input to the class having most examples among the k neighbors of the input. All neighbors have equal vote, and the class having the maximum number of voters among the k neighbors is chosen. Ties are broken arbitrarily or a weighted vote is taken. k is generally taken to be an odd number to minimize ties: Confusion is

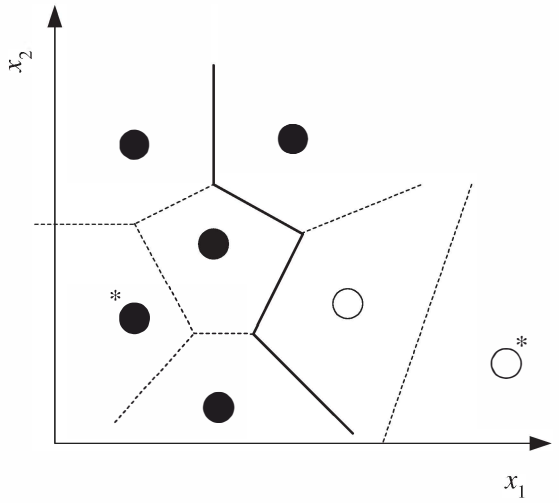

Figure 8.5 Dotted lines are the Voronoi tesselation and the straight line is the class discriminant. In condensed nearest neighbor, those instances that do not participate in defining the discriminant (marked by '*') can be removed without increasing the training error.

NEAREST NEIGHBOR CLASSIFIER

VORONOI TESSELATION

generally between two neighboring classes. A special case of k-nn is the *nearest neighbor classifier* where $k = 1$ and the input is assigned to the class of the nearest pattern. This divides the space in the form of a *Voronoi tesselation* (see figure 8.5).

8.5 Condensed Nearest Neighbor

Time and space complexity of nonparametric methods are proportional to the size of the training set, and *condensing* methods have been proposed to decrease the number of stored instances without degrading performance. The idea is to select the smallest subset $\mathcal{Z}$ of $\mathcal{X}$ such that when $\mathcal{Z}$ is used in place of $\mathcal{X}$, error does not increase (Dasarathy 1991).

CONDENSED NEAREST NEIGHBOR

The best-known and earliest method is *condensed nearest neighbor* where 1-nn is used as the nonparametric estimator for classification (Hart 1968). 1-nn approximates the discriminant in a piecewise linear manner, and only the instances that define the discriminant need be kept; an in-

$\mathcal{Z} \leftarrow \emptyset$
Repeat
  For all $\boldsymbol{x} \in \mathcal{X}$ (in random order)
    Find $\boldsymbol{x}' \in \mathcal{Z}$ such that $\|\boldsymbol{x} - \boldsymbol{x}'\| = \min_{\boldsymbol{x}^j \in \mathcal{Z}} \|\boldsymbol{x} - \boldsymbol{x}^j\|$
    If class($\boldsymbol{x}$)$\neq$class($\boldsymbol{x}'$) add $\boldsymbol{x}$ to $\mathcal{Z}$
Until $\mathcal{Z}$ does not change

Figure 8.6 Condensed nearest neighbor algorithm.

stance inside the class regions need not be stored as *its* nearest neighbor is of the same class and its absence does not cause any error (on the training set) (figure 8.5). Such a subset is called a consistent subset, and we would like to find the minimal consistent subset.

Hart proposed a greedy algorithm to find $\mathcal{Z}$ (figure 8.6). The algorithm starts with an empty $\mathcal{Z}$ and passing over the instances in $\mathcal{X}$ one by one in a random order, checks whether they can be classified correctly by 1-nn using the instances already stored in $\mathcal{Z}$. If an instance is misclassified, it is added to $\mathcal{Z}$; if it is correctly classified, $\mathcal{Z}$ is unchanged. We should pass over the training set a few times until no further instances are added. The algorithm does a local search and depending on the order in which the training instances are seen, different subsets may be found, which may have different accuracies on the validation data. Thus it does not guarantee finding the minimal consistent subset, which is known to be NP-complete (Wilfong 1992).

Condensed nearest neighbor is a greedy algorithm that aims to minimize training error and complexity, measured by the size of the stored subset. We can write an augmented error function

$$E'(\mathcal{Z}|\mathcal{X}) = E(\mathcal{X}|\mathcal{Z}) + \lambda|\mathcal{Z}| \tag{8.17}$$

where $E(\mathcal{X}|\mathcal{Z})$ is the error on $\mathcal{X}$ storing $\mathcal{Z}$. $|\mathcal{Z}|$ is the cardinality of $\mathcal{Z}$, and the second term penalizes complexity. As in any regularization scheme, λ represents the trade-off between the error and complexity such that for small λ, error becomes more important, and as λ gets larger, complex models are penalized more. Condensed nearest neighbor is one method to minimize equation 8.17, but other algorithms to optimize it can also be devised.

8.6 Distance-Based Classification

The k-nearest neighbor classifier assigns an instance to the class most heavily represented among its neighbors. It is based on the idea that the more similar the instances, the more likely it is that they belong to the same class. We can use the same approach for classification as long as we have a reasonable similarity or distance measure (Chen et al. 2009).

Most classification algorithms can be recast as a distance-based classifier. For example, in section 5.5, we saw the parametric approach with Gaussian classes, and there, we talked about the *nearest mean classifier* where we choose C_i if

$$\mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_i) = \min_{j=1}^{K} \mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_j) \tag{8.18}$$

In the case of hyperspheric Gaussians where dimensions are independent and all are in the same scale, the distance measure is the Euclidean:

$$\mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_i) = \|\boldsymbol{x} - \boldsymbol{m}_i\|$$

Otherwise it is the Mahalanobis distance:

$$\mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_i) = (\boldsymbol{x} - \boldsymbol{m}_i)^T \mathbf{S}_i^{-1} (\boldsymbol{x} - \boldsymbol{m}_i)$$

where $\mathbf{S}_i$ is the covariance matrix of C_i.

In the semiparametric approach where each class is written as a mixture of Gaussians, we can say roughly speaking that we choose C_i if among all cluster centers of all classes, one that belongs to C_i is the closest:

$$\min_{l=1}^{k_i} \mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_{il}) = \min_{j=1}^{K} \min_{l=1}^{k_j} \mathcal{D}(\boldsymbol{x}, \boldsymbol{m}_{jl}) \tag{8.19}$$

where k_j is the number of clusters of C_j and $\boldsymbol{m}_{jl}$ denotes the center of cluster l of C_j. Again, the distance used is the Euclidean or Mahalanobis depending on the shape of the clusters.

The nonparametric case can be even more flexible: Instead of having a distance measure per class or per cluster, we can have a different one for each neighborhood, that is, for each small region in the input space. In other words, we can define *locally adaptive distance functions* that we can then use in classification, for example, with k-nn (Hastie and Tibshirani 1996; Domeniconi, Peng, and Gunopulos 2002; Ramanan and Baker 2011).

DISTANCE LEARNING

The idea of *distance learning* is to parameterize $\mathcal{D}(\boldsymbol{x}, \boldsymbol{x}^t|\theta)$, learn θ from a labeled sample in a supervised manner, and then use it with k-nn (Bellet, Habrard, and Sebban 2013). The most common approach is to use the Mahalanobis distance:

$$\mathcal{D}(\boldsymbol{x}, \boldsymbol{x}^t|\mathbf{M}) = (\boldsymbol{x} - \boldsymbol{x}^t)^T \mathbf{M} (\boldsymbol{x} - \boldsymbol{x}^t) \tag{8.20}$$

where the parameter is the positive definite matrix $\mathbf{M}$. An example is the *large margin nearest neighbor* algorithm (Weinberger and Saul 2009) where $\mathbf{M}$ is estimated so that for all instances in the training set, the distance to a neighbor with the same label is always less than the distance to a neighbor with a different label—we discuss this algorithm in detail in section 13.13.

LARGE MARGIN NEAREST NEIGHBOR

When the input dimensionality is high, to avoid overfitting, one approach is to add sparsity constraints on $\mathbf{M}$. The other approach is to use a low-rank approximation where we factor $\mathbf{M}$ as $\mathbf{L}^T\mathbf{L}$ and $\mathbf{L}$ is $k \times d$ with $k < d$. In this case:

$$\begin{aligned} \mathcal{D}(\boldsymbol{x}, \boldsymbol{x}^t|\mathbf{M}) &= (\boldsymbol{x} - \boldsymbol{x}^t)^T \mathbf{M} (\boldsymbol{x} - \boldsymbol{x}^t) = (\boldsymbol{x} - \boldsymbol{x}^t)^T \mathbf{L}^T \mathbf{L} (\boldsymbol{x} - \boldsymbol{x}^t) \\ &= (\mathbf{L}(\boldsymbol{x} - \boldsymbol{x}^t))^T (\mathbf{L}(\boldsymbol{x} - \boldsymbol{x}^t)) = (\mathbf{L}\boldsymbol{x} - \mathbf{L}\boldsymbol{x}^t)^T (\mathbf{L}\boldsymbol{x} - \mathbf{L}\boldsymbol{x}^t)) \\ &= (\boldsymbol{z} - \boldsymbol{z}^t)^T (\boldsymbol{z} - \boldsymbol{z}^t) = \|\boldsymbol{z} - \boldsymbol{z}^t\|^2 \end{aligned} \tag{8.21}$$

where $\boldsymbol{z} = \mathbf{L}\boldsymbol{x}$ is the k-dimensional projection of $\boldsymbol{x}$, and we learn $\mathbf{L}$ instead of $\mathbf{M}$. We see that the Mahalanobis distance in the original d-dimensional $\boldsymbol{x}$ space corresponds to the (squared) Euclidean distance in the new k-dimensional space. This implies the three-way relationship between distance estimation, dimensionality reduction, and feature extraction: The ideal distance measure is defined as the Euclidean distance in a new space whose (fewest) dimensions are extracted from the original inputs in the best possible way. This is demonstrated in figure 8.7.

HAMMING DISTANCE

With discrete data, *Hamming distance* that counts the number of nonmatching attributes can be used:

$$HD(\boldsymbol{x}, \boldsymbol{x}^t) = \sum_{j=1}^{d} 1(x_j \neq x_j^t) \tag{8.22}$$

where

$$1(a) = \begin{cases} 1 & \text{if } a \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

This framework can be used with application-dependent similarity or distance measures as well. We may have specialized similarity/distance

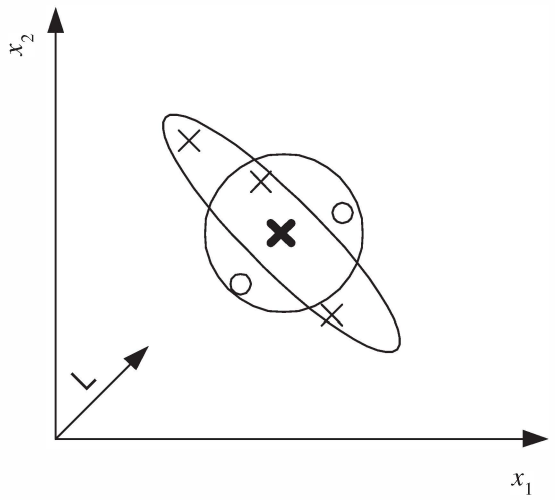

Figure 8.7 The use of Mahalanobis vs. Euclidean distance in k-nearest neighbor classification. There are two classes indicated by '$\circ$' and '$\times$'. The bold '$\times$' is the test instance and $k = 3$. Points that are of equal Euclidean distance define a circle that here leads to misclassification. We see that there is a certain correlation structure that can be captured by the Mahalanobis distance; it defines an ellipse and leads to correct classification. We also see that if the data is projected on the direction showed by L, we can do correct classification in that reduced one-dimensional space.

scores for matching image parts in vision, sequence alignment scores in bioinformatics, and document similarity measures in natural language processing; these can all be used without explicitly needing to represent those entities as vectors and using a general-purpose distance such as the Euclidean distance. In chapter 13, we will talk about kernel functions that have a similar role.

As long as we have a similarity score function between two instances $S(\boldsymbol{x}, \boldsymbol{x}^t)$, we can define a *similarity-based representation* $\boldsymbol{x}'$ of instance $\boldsymbol{x}$ as the N-dimensional vector of scores with all the training instances, $\boldsymbol{x}^t, t = 1, \ldots, N$:

$$\boldsymbol{x}' = [s(\boldsymbol{x}, \boldsymbol{x}^1), s(\boldsymbol{x}, \boldsymbol{x}^2), \ldots, s(\boldsymbol{x}, \boldsymbol{x}^N)]^T$$

This can then be used as a vector to be handled by any learner (Pekalska

and Duin 2002); in the context of kernel machines, we will call this the *empirical kernel map* (section 13.7).

8.7 Outlier Detection

An *outlier*, *novelty*, or *anomaly* is an instance that is very much different from other instances in the sample. An outlier may indicate an abnormal behavior of the system; for example, in a dataset of credit card transactions, it may indicate fraud; in an image, outliers may indicate anomalies, for example, tumors; in a dataset of network traffic, outliers may be intrusion attempts; in a health-care scenario, an outlier indicates a significant deviation from patient's normal behavior. Outliers may also be recording errors—for example, due to faulty sensors—that should be detected and discarded to get reliable statistics.

OUTLIER DETECTION

Outlier detection is not generally cast as a supervised, two-class classification problem of seperating typical instances and outliers, because generally there are very few instances that can be labeled as outliers and they do not fit a consistent pattern that can be easily captured by a two-class classifier. Instead, it is the typical instances that are modeled; this is sometimes called *one-class classification*. Once we model the typical instances, any instance that does not fit the model (and this may occur in many different ways) is an anomaly. Another problem that generally occurs is that the data used to train the outlier detector is unlabeled and may contain outliers mixed with typical instances.

ONE-CLASS CLASSIFICATION

Outlier detection basically implies spotting what does not normally happen; that is, it is density estimation followed by checking for instances with too small probability under the estimated density. As usual, the fitted model can be parametric, semiparametric, or nonparametric. In the parametric case (section 5.4), for example, we can fit a Gaussian to the whole data and any instance having a low probability, or equally, with high Mahalanobis distance to the mean, is a candidate for being an outlier. In the semiparametric case (section 7.2), we fit, for example, a mixture of Gaussians and check whether an instance has small probability; this would be an instance that is far from its nearest cluster center or one that forms a cluster by itself.

Still when the data that is used for fitting the model itself includes outliers, it makes more sense to use a nonparametric density estimator, because the more parametric a model is, the less robust it will be to the

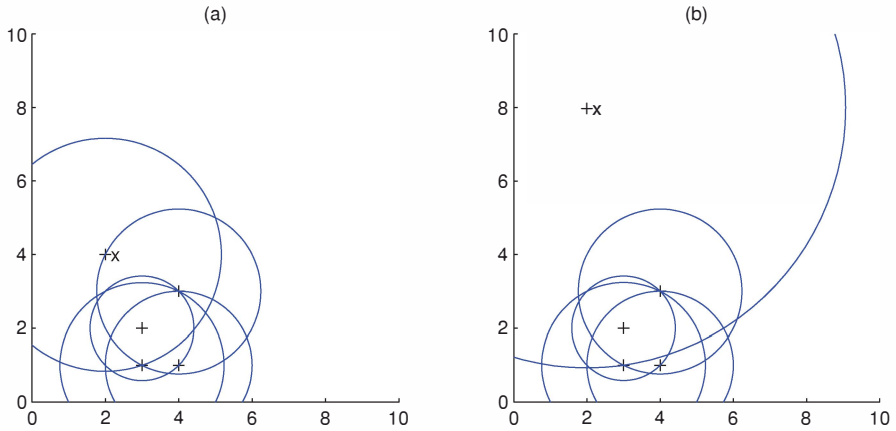

Figure 8.8 Training instances are shown by '+', '×' is the query, and the radius of the circle centered at an instance is equal to the distance to the third nearest neighbor. (a) LOF of '×' is close to 1 and it is not an outlier. (b) LOF of '×' is much larger than 1 and it is likely to be an outlier.

presence of outliers—for example, a single outlier may seriously corrupt the estimated mean and covariance of a Gaussian.

In nonparametric density estimation, as we discussed in the preceding sections, the estimated probability is high where there are many training instances nearby and the probability decreases as the neighborhood becomes more sparse. One example is the *local outlier factor* that compares the denseness of the neighborhood of an instance with the average denseness of the neighborhoods of its neighbors (Breunig et al. 2000). Let us define $d_k(\boldsymbol{x})$ as the distance between instance $\boldsymbol{x}$ and its k-th nearest neighbor. Let us define $\mathcal{N}(\boldsymbol{x})$ as the set of training instances that are in the neighborhood of $\boldsymbol{x}$, for example, its k nearest neighbors. Consider $d_k(\boldsymbol{s})$ for $\boldsymbol{s} \in \mathcal{N}(\boldsymbol{x})$. We compare $d_k(\boldsymbol{x})$ with the average of $d_k(\boldsymbol{s})$ for such $\boldsymbol{s}$:

LOCAL OUTLIER FACTOR

$$\text{LOF}(\boldsymbol{x}) = \frac{d_k(\boldsymbol{x})}{\sum_{\boldsymbol{s} \in \mathcal{N}(\boldsymbol{x})} d_k(\boldsymbol{s}) / |\mathcal{N}(\boldsymbol{x})|} \tag{8.23}$$

If $\text{LOF}(\boldsymbol{x})$ is close to 1, $\boldsymbol{x}$ is not an outlier; as it gets larger, the probability that it is an outlier increases (see figure 8.8).

8.8 Nonparametric Regression: Smoothing Models

In regression, given the training set $\mathcal{X} = \{x^t, r^t\}$ where $r^t \in \Re$, we assume

$$r^t = g(x^t) + \epsilon$$

In parametric regression, we assume a polynomial of a certain order and compute its coefficients that minimize the sum of squared error on the training set. Nonparametric regression is used when no such polynomial can be assumed; we only assume that close x have close $g(x)$ values. As in nonparametric density estimation, given x, our approach is to find the neighborhood of x and average the r values in the neighborhood to calculate $\hat{g}(x)$. The nonparametric regression estimator is also called a *smoother* and the estimate is called a *smooth* (Härdle 1990). There are various methods for defining the neighborhood and averaging in the neighborhood, similar to methods in density estimation. We discuss the methods for the univariate x; they can be generalized to the multivariate case in a straightforward manner using multivariate kernels, as in density estimation.

SMOOTHER

8.8.1 Running Mean Smoother

If we define an origin and a bin width and average the r values in the bin as in the histogram, we get a *regressogram* (see figure 8.9)

REGRESSOGRAM

$$\hat{g}(x) = \frac{\sum_{t=1}^{N} b(x, x^t) r^t}{\sum_{t=1}^{N} b(x, x^t)} \tag{8.24}$$

where

$$b(x, x^t) = \begin{cases} 1 & \text{if } x^t \text{ is the same bin with } x \\ 0 & \text{otherwise} \end{cases}$$

Having discontinuities at bin boundaries is disturbing as is the need to fix an origin. As in the naive estimator, in the *running mean smoother*, we define a bin symmetric around x and average in there (figure 8.10).

RUNNING MEAN SMOOTHER

$$\hat{g}(x) = \frac{\sum_{t=1}^{N} w\left(\frac{x - x^t}{h}\right) r^t}{\sum_{t=1}^{N} w\left(\frac{x - x^t}{h}\right)} \tag{8.25}$$

where

$$w(u) = \begin{cases} 1 & \text{if } |u| < 1 \\ 0 & \text{otherwise} \end{cases}$$

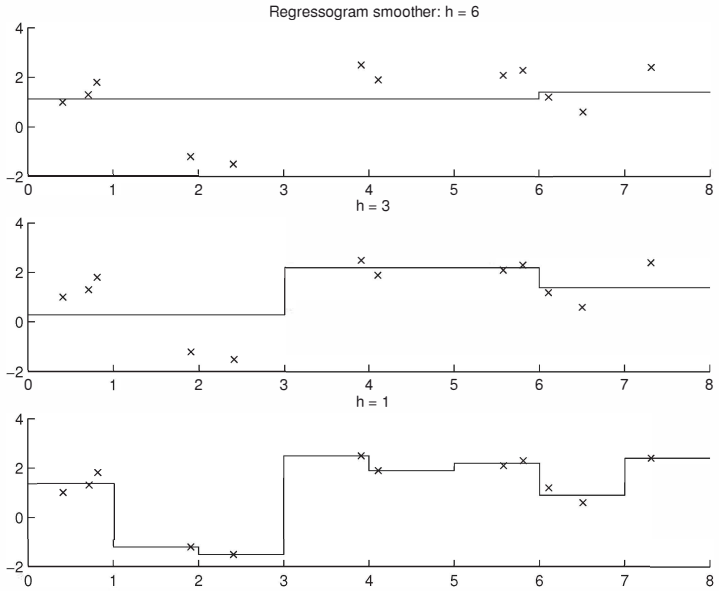

Figure 8.9 Regressograms for various bin lengths. '×' denote data points.

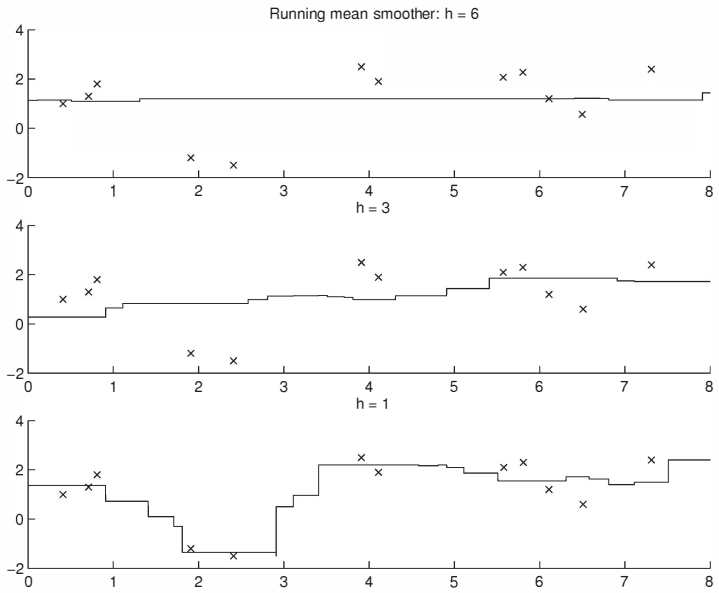

Figure 8.10 Running mean smooth for various bin lengths.

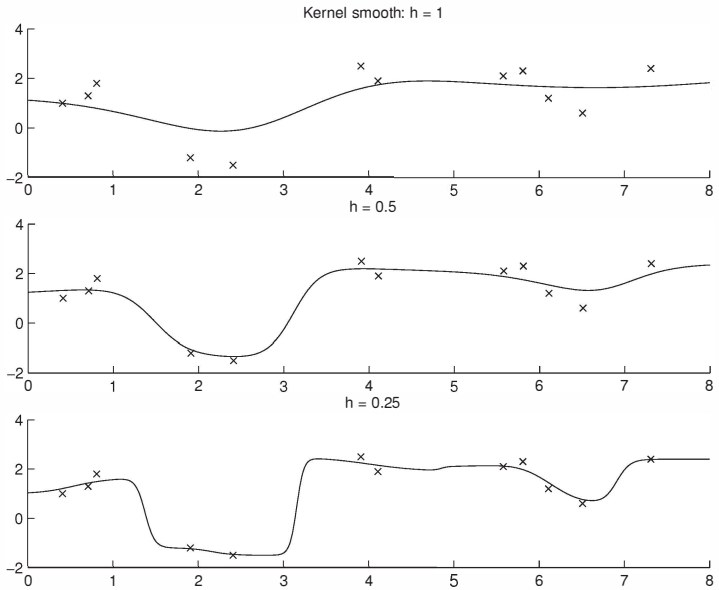

Figure 8.11 Kernel smooth for various bin lengths.

This method is especially popular with evenly spaced data, such as time series. In applications where there is noise, we can use the median of the r^t in the bin instead of their mean.

8.8.2 Kernel Smoother

KERNEL SMOOTHER

As in the kernel estimator, we can use a kernel giving less weight to further points, and we get the *kernel smoother* (see figure 8.11):

$$\hat{g}(x) = \frac{\sum_t K\left(\frac{x-x^t}{h}\right) r^t}{\sum_t K\left(\frac{x-x^t}{h}\right)} \tag{8.26}$$

k-NN SMOOTHER

Typically a Gaussian kernel $K(\cdot)$ is used. Instead of fixing h, we can fix k, the number of neighbors, adapting the estimate to the density around x, and get the *k-nn smoother*.

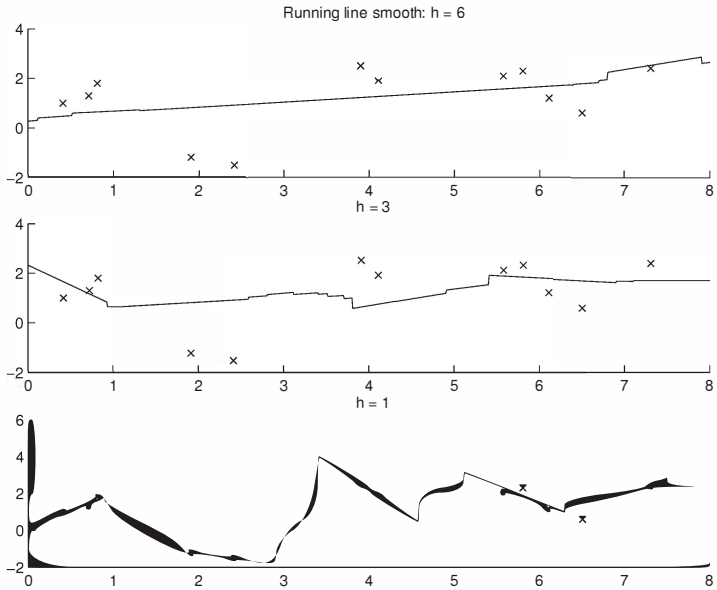

Figure 8.12 Running line smooth for various bin lengths.

8.8.3 Running Line Smoother

Instead of taking an average and giving a constant fit at a point, we can take into account one more term in the Taylor expansion and calculate a linear fit. In the *running line smoother*, we can use the data points in the neighborhood, as defined by h or k, and fit a local regression line (see figure 8.12).

RUNNING LINE SMOOTHER

In the *locally weighted running line smoother*, known as *loess*, instead of a hard definition of neighborhoods, we use kernel weighting such that distant points have less effect on error.

LOCALLY WEIGHTED RUNNING LINE SMOOTHER

8.9 How to Choose the Smoothing Parameter

In nonparametric methods, for density estimation or regression, the critical parameter is the smoothing parameter as used in bin width or kernel spread h, or the number of neighbors k. The aim is to have an estimate that is less variable than the data points. As we have discussed previously, one source of variability in the data is noise and the other is the

variability in the unknown underlying function. We should smooth just enough to get rid of the effect of noise—not less, not more. With too large h or k, many instances contribute to the estimate at a point and we also smooth the variability due to the function (oversmoothing); with too small h or k, single instances have a large effect and we do not even smooth over the noise (undersmoothing). In other words, small h or k leads to small bias but large variance. Larger h or k decreases variance but increases bias. Geman, Bienenstock, and Doursat (1992) discuss bias and variance for nonparametric estimators.

SMOOTHING SPLINES

This requirement is explicitly coded in a regularized cost function as used in *smoothing splines*:

$$\sum_t \left[r^t - \hat{g}(x^t)\right]^2 + \lambda \int_a^b [\hat{g}''(x)]^2 dx \tag{8.27}$$

The first term is the error of fit. $[a, b]$ is the input range; $\hat{g}''(\cdot)$ is the *curvature* of the estimated function $\hat{g}(\cdot)$ and as such measures the variability. Thus the second term penalizes fast-varying estimates. λ trades off variability and error where, for example, with large λ, we get smoother estimates.

Cross-validation is used to tune h, k, or λ. In density estimation, we choose the parameter value that maximizes the likelihood of the validation set. In a supervised setting, trying a set of candidates on the training set (see figure 8.13), we choose the parameter value that minimizes the error on the validation set.

8.10 Notes

k-nearest neighbor and kernel-based estimation were proposed sixty years ago, but because of the need for large memory and computation, the approach was not popular for a long time (Aha, Kibler, and Albert 1991). With advances in parallel processing and with memory and computation getting cheaper, such methods have recently become more widely used. Textbooks on nonparametric estimation are Silverman 1986 and Scott 1992. Dasarathy 1991 is a collection of many papers on k-nn and editing/condensing rules; Aha 1997 is another collection.

The nonparametric methods are very easy to parallelize on a Single Instruction Multiple Data (SIMD) machine; each processor stores one training instance in its local memory and in parallel computes the kernel

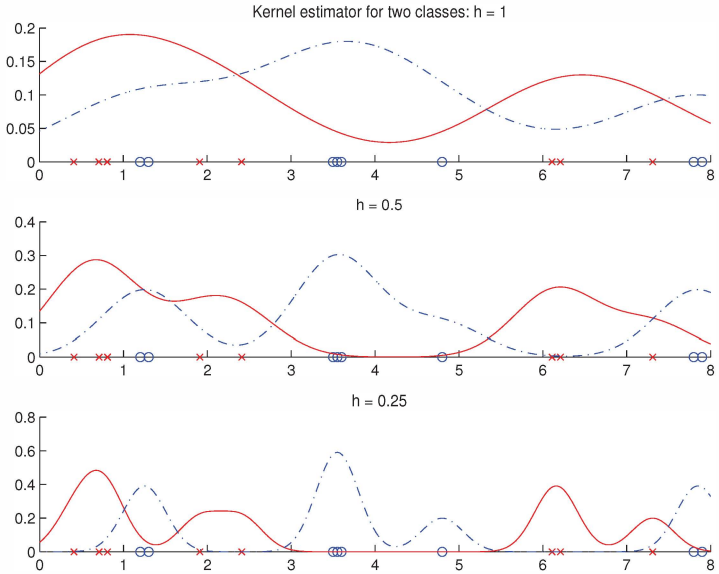

Figure 8.13 Kernel estimate for various bin lengths for a two-class problem. Plotted are the conditional densities, $p(x|C_i)$. It seems that the top one oversmooths and the bottom undersmooths, but whichever is the best will depend on where the validation data points are.

function value for that instance (Stanfill and Waltz 1986). Multiplying with a kernel function can be seen as a convolution, and we can use Fourier transformation to calculate the estimate more efficiently (Silverman 1986). It has also been shown that spline smoothing is equivalent to kernel smoothing.

CASE-BASED REASONING

In artificial intelligence, the nonparametric approach is called *case-based reasoning*. The output is found by interpolating from known similar past "cases." This also allows for some knowledge extraction: The given output can be justified by listing these similar past cases.

Due to its simplicity, k-nn is the most widely used nonparametric classification method and is quite successful in practice in a variety of applications. One nice property is that they can be used even with very few labeled instances; for example, in a forensic application, we may have only one face image per person.

It has been shown (Cover and Hart 1967; reviewed in Duda, Hart, and

Stork 2001) that in the large sample case when $N \to \infty$, the risk of nearest neighbor ($k = 1$) is never worse than twice the Bayes' risk (which is the best that can be achieved), and, in that respect, it is said that "half of the available information in an infinite collection of classified samples is contained in the nearest neighbor" (Cover and Hart 1967, 21). In the case of k-nn, it has been shown that the risk asymptotes to the Bayes' risk as k goes to infinity.

The most critical factor in nonparametric estimation is the distance metric used. With discrete attributes, we can simply use the Hamming distance where we just sum up the number of nonmatching attributes. More sophisticated distance functions are discussed in Wettschereck, Aha, and Mohri 1997 and Webb 1999.

Distance estimation or metric learning is a popular research area; see Bellet, Habrard, and Sebban 2013 for a comprehensive recent survey. The different ways similarity measures can be used in classification are discussed by Chen et al. (2009); examples of local distance methods in computer vision are given in Ramanan and Baker 2011.

Outlier/anomaly/novelty detection arises as an interesting problem in various contexts, from faults to frauds, and in detecting significant deviations from the past data, for example, churning customers. It is a very popular research area, and two comprehensive surveys include those by Hodge and Austin (2004) and Chandola, Banerjee, and Kumar (2009).

ADDITIVE MODELS

Nonparametric regression is discussed in detail in Härdle 1990. Hastie and Tibshirani (1990) discuss smoothing models and propose *additive models* where a multivariate function is written as a sum of univariate estimates. Locally weighted regression is discussed in Atkeson, Moore, and Schaal 1997. These models bear much similarity to radial basis functions and mixture of experts that we discuss in chapter 12.

In the condensed nearest neighbor algorithm, we saw that we can keep only a subset of the training instances, those that are close to the boundary, and we can define the discriminant using them only. This idea bears much similarity to the *support vector machines* that we discuss in chapter 13. There we also discuss various kernel functions to measure similarity between instances and how we can choose the best. Writing the prediction as a sum of the combined effects of training instances also underlies *Gaussian processes* (chapter 16), where a kernel function is called a *covariance function*.

8.11 Exercises

1. How can we have a smooth histogram?

   SOLUTION: We can interpolate between the two nearest bin centers. We can consider the bin centers as x^t, consider the histogram values as r^t, and use any interpolation scheme, linear or kernel-based.

2. Show equation 8.16.

   SOLUTION: Given that

   $$\hat{p}(\boldsymbol{x}|C_i) = \frac{k_i}{N_i V^k(\boldsymbol{x})} \quad \text{and} \quad \hat{P}(C_i) = \frac{N_i}{N}$$

   we can write

   $$\begin{aligned}\hat{P}(C_i|\boldsymbol{x}) &= \frac{\hat{p}(\boldsymbol{x}|C_i)\hat{P}(C_i)}{\sum_j \hat{p}(\boldsymbol{x}|C_j)\hat{P}(C_j)} = \frac{\frac{k_i}{N_i V^k(\boldsymbol{x})}\frac{N_i}{N}}{\sum_j \frac{k_j}{N_j V^k(\boldsymbol{x})}\frac{N_j}{N}} \\ &= \frac{k_i}{\sum_j k_j} = \frac{k_i}{k}\end{aligned}$$

3. Parametric regression (section 5.8) assumes Gaussian noise and hence is not robust to outliers; how can we make it more robust ?

4. How can we detect outliers after hierarchical clustering (section 7.8) ?

5. How does condensed nearest neighbor behave if $k > 1$?

   SOLUTION: When $k > 1$, to get full accuracy without any misclassification, it may be necessary to store an instance multiple times so that the correct class gets the majority of the votes. For example, if $k = 3$ and $\boldsymbol{x}$ has two neighbors both belonging to a different class, we need to store $\boldsymbol{x}$ twice (i.e., it gets added in two epochs), so that if $\boldsymbol{x}$ is seen during test, the majority (two in this case) out of three neighbors belong to the correct class.

6. In condensed nearest neighbor, an instance previously added to $\mathcal{Z}$ may no longer be necessary after a later addition. How can we find such instances that are no longer necessary?

7. In a regressogram, instead of averaging in a bin and doing a constant fit, we can use the instances falling in a bin and do a linear fit (see figure 8.14). Write the code and compare this with the regressogram proper.

8. Write the error function for loess discussed in section 8.8.3.

   SOLUTION: The output is calculated using a linear model $g(x) = ax+b$, where, in the running line smoother, we minimize

   $$E(a, b|x, \mathcal{X}) = \sum_t w\left(\frac{x - x^t}{h}\right)[r^t - (ax^t + b)]^2$$

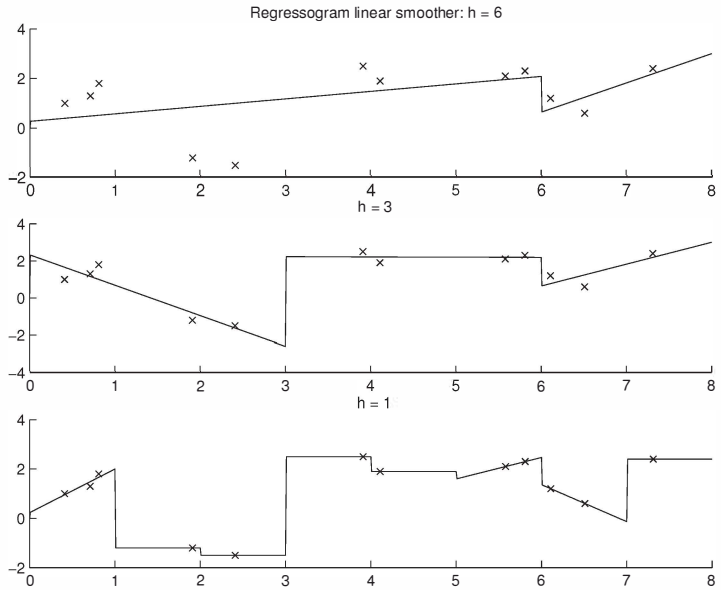

Figure 8.14 Regressograms with linear fits in bins for various bin lengths.

and

$$w(u) = \begin{cases} 1 & \text{if } |u| < 1 \\ 0 & \text{otherwise} \end{cases}$$

Note that we do not have one error function but rather, for each test input x, we have another error function taking into account only the data closest to x, which is minimized to fit a line in that neighborhood.

Loess is the weighted version of running line smoother where a kernel function $K(\cdot) \in (0,1)$ replaces the $w(\cdot) \in \{0,1\}$:

$$E(a,b|x,\mathcal{X}) = \sum_t K\left(\frac{x-x^t}{h}\right)[r^t - (ax^t+b)]^2$$

9. Propose an incremental version of the running mean estimator, which, like the condensed nearest neighbor, stores instances only when necessary.
10. Generalize kernel smoother to multivariate data.
11. In the running smoother, we can fit a constant, a line, or a higher-degree polynomial at a test point. How can we choose among them?

    SOLUTION: By cross-validation.

12. In the running mean smoother, besides giving an estimate, can we also calculate a confidence interval indicating the variance (uncertainty) around the estimate at that point?

8.12 References

Aha, D. W., ed. 1997. Special Issue on Lazy Learning. *Artificial Intelligence Review* 11 (1–5): 7–423.

Aha, D. W., D. Kibler, and M. K. Albert. 1991. "Instance-Based Learning Algorithm." *Machine Learning* 6:37–66.

Atkeson, C. G., A. W. Moore, and S. Schaal. 1997. "Locally Weighted Learning." *Artificial Intelligence Review* 11:11–73.

Bellet, A., A. Habrard, and M. Sebban. 2013. "A Survey on Metric Learning for Feature Vectors and Structured Data." *arXiv:1306.6709v2*.

Breunig, M. M., H.-P. Kriegel, R. T. Ng, and J. Sander. 2000. "LOF: Identifying Density-Based Local Outliers." In *ACM SIGMOD International Conference on Management of Data*, 93–104. New York: ACM Press.

Chandola, V., A. Banerjee, and V. Kumar. 2009. "Anomaly Detection: A Survey." *ACM Computing Surveys* 41 (3): 15:1–15:58.

Chen, Y., E. K. Garcia, M. R. Gupta, A. Rahimi, and L. Cazzanti. 2009. "Similarity-Based Classification: Concepts and Algorithms." *Journal of Machine Learning Research* 11:747–776.

Cover, T. M., and P. E. Hart. 1967. "Nearest Neighbor Pattern Classification." *IEEE Transactions on Information Theory* 13:21–27.

Dasarathy, B. V. 1991. *Nearest Neighbor Norms: NN Pattern Classification Techniques*. Los Alamitos, CA: IEEE Computer Society Press.

Domeniconi, C., J. Peng, and D. Gunopulos. 2002. "Locally Adaptive Metric Nearest-Neighbor Classification." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24:1281–1285.

Duda, R. O., P. E. Hart, and D. G. Stork. 2001. *Pattern Classification*, 2nd ed. New York: Wiley.

Geman, S., E. Bienenstock, and R. Doursat. 1992. "Neural Networks and the Bias/Variance Dilemma." *Neural Computation* 4:1–58.

Härdle, W. 1990. *Applied Nonparametric Regression*. Cambridge, UK: Cambridge University Press.

Hart, P. E. 1968. "The Condensed Nearest Neighbor Rule." *IEEE Transactions on Information Theory* 14:515–516.

Hastie, T. J., and R. J. Tibshirani. 1990. *Generalized Additive Models*. London: Chapman and Hall.

Hastie, T. J., and R. J. Tibshirani. 1996. "Discriminant Adaptive Nearest Neighbor Classification." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18:607–616.

Hodge, V. J., and J. Austin. 2004. "A Survey of Outlier Detection Methodologies." *Artificial Intelligence Review* 22:85–126.

Pekalska, E., and R. P. W. Duin. 2002. "Dissimilarity Representations Allow for Building Good Classifiers." *Pattern Recognition Letters* 23:943–956.

Ramanan, D., and S. Baker. 2011. "Local Distance Functions: A Taxonomy, New Algorithms, and an Evaluation." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33:794–806.

Scott, D. W. 1992. *Multivariate Density Estimation*. New York: Wiley.

Silverman, B. W. 1986. *Density Estimation in Statistics and Data Analysis*. London: Chapman and Hall.

Stanfill, C., and D. Waltz. 1986. "Toward Memory-Based Reasoning." *Communications of the ACM* 29:1213–1228.

Webb, A. 1999. *Statistical Pattern Recognition*. London: Arnold.

Weinberger, K. Q., and L. K. Saul. 2009. "Distance Metric Learning for Large Margin Classification." *Journal of Machine Learning Research* 10:207–244.

Wettschereck, D., D. W. Aha, and T. Mohri. 1997. "A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms." *Artificial Intelligence Review* 11:273–314.

Wilfong, G. 1992. "Nearest Neighbor Problems." *International Journal on Computational Geometry and Applications* 2:383–416.