variable X_j. Since all vectors are assumed to be column vectors, the ith row of $\mathbf{X}$ is x_i^T, the vector transpose of x_i.

For the moment we can loosely state the learning task as follows: given the value of an input vector X, make a good prediction of the output Y, denoted by $\hat{Y}$ (pronounced "y-hat"). If Y takes values in $\mathbb{R}$ then so should $\hat{Y}$; likewise for categorical outputs, $\hat{G}$ should take values in the same set $\mathcal{G}$ associated with G.

For a two-class G, one approach is to denote the binary coded target as Y, and then treat it as a quantitative output. The predictions $\hat{Y}$ will typically lie in $[0, 1]$, and we can assign to $\hat{G}$ the class label according to whether $\hat{y} > 0.5$. This approach generalizes to K-level qualitative outputs as well.

We need data to construct prediction rules, often a lot of it. We thus suppose we have available a set of measurements (x_i, y_i) or (x_i, g_i), $i = 1, \ldots, N$, known as the *training data*, with which to construct our prediction rule.

2.3 Two Simple Approaches to Prediction: Least Squares and Nearest Neighbors

In this section we develop two simple but powerful prediction methods: the linear model fit by least squares and the k-nearest-neighbor prediction rule. The linear model makes huge assumptions about structure and yields stable but possibly inaccurate predictions. The method of k-nearest neighbors makes very mild structural assumptions: its predictions are often accurate but can be unstable.

2.3.1 Linear Models and Least Squares

The linear model has been a mainstay of statistics for the past 30 years and remains one of our most important tools. Given a vector of inputs $X = (X_1, X_2, \ldots, X_p)$, we predict the output Y via the model

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j. \tag{2.1}$$

The term $\hat{\beta}_0$ is the intercept, also known as the *bias* in machine learning. Often it is convenient to include the constant variable 1 in X, include $\hat{\beta}_0$ in the vector of coefficients $\hat{\beta}$, and then write the linear model in vector form as an inner product

$$\hat{Y} = X^T \hat{\beta}, \tag{2.2}$$

where X^T denotes vector or matrix transpose (X being a column vector). Here we are modeling a single output, so $\hat{Y}$ is a scalar; in general $\hat{Y}$ can be a K–vector, in which case β would be a $p \times K$ matrix of coefficients. In the $(p+1)$-dimensional input–output space, $(X, \hat{Y})$ represents a hyperplane. If the constant is included in X, then the hyperplane includes the origin and is a subspace; if not, it is an affine set cutting the Y-axis at the point $(0, \hat{\beta}_0)$. From now on we assume that the intercept is included in $\hat{\beta}$.

Viewed as a function over the p-dimensional input space, $f(X) = X^T\beta$ is linear, and the gradient $f'(X) = \beta$ is a vector in input space that points in the steepest uphill direction.

How do we fit the linear model to a set of training data? There are many different methods, but by far the most popular is the method of *least squares*. In this approach, we pick the coefficients β to minimize the residual sum of squares

$$\text{RSS}(\beta) = \sum_{i=1}^{N} (y_i - x_i^T\beta)^2. \tag{2.3}$$

$\text{RSS}(\beta)$ is a quadratic function of the parameters, and hence its minimum always exists, but may not be unique. The solution is easiest to characterize in matrix notation. We can write

$$\text{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta), \tag{2.4}$$

where $\mathbf{X}$ is an $N \times p$ matrix with each row an input vector, and $\mathbf{y}$ is an N-vector of the outputs in the training set. Differentiating w.r.t. β we get the *normal equations*

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0. \tag{2.5}$$

If $\mathbf{X}^T\mathbf{X}$ is nonsingular, then the unique solution is given by

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}, \tag{2.6}$$

and the fitted value at the ith input x_i is $\hat{y}_i = \hat{y}(x_i) = x_i^T\hat{\beta}$. At an arbitrary input x_0 the prediction is $\hat{y}(x_0) = x_0^T\hat{\beta}$. The entire fitted surface is characterized by the p parameters $\hat{\beta}$. Intuitively, it seems that we do not need a very large data set to fit such a model.

Let's look at an example of the linear model in a classification context. Figure 2.1 shows a scatterplot of training data on a pair of inputs X_1 and X_2. The data are simulated, and for the present the simulation model is not important. The output class variable G has the values GREEN or RED, and is represented as such in the scatterplot. There are 100 points in each of the two classes. The linear regression model was fit to these data, with the response Y coded as 0 for GREEN and 1 for RED. The fitted values $\hat{Y}$ are

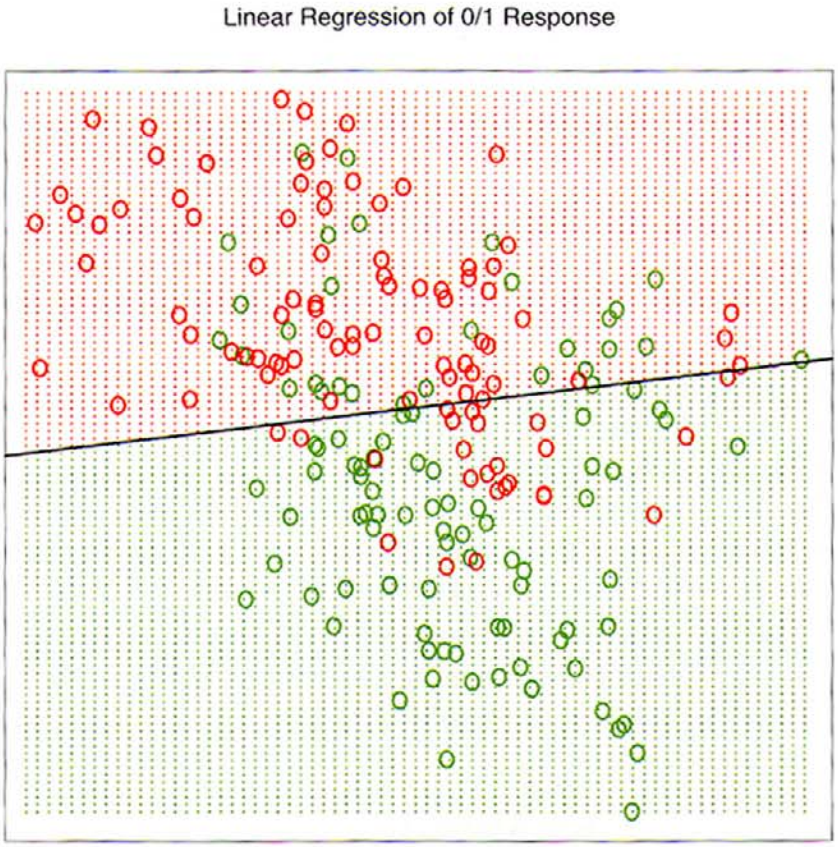

FIGURE 2.1. *A classification example in two dimensions. The classes are coded as a binary variable—*GREEN $= 0$, RED $= 1$*—and then fit by linear regression. The line is the decision boundary defined by* $x^T\hat{\beta} = 0.5$*. The red shaded region denotes that part of input space classified as* RED*, while the green region is classified as* GREEN*.*

converted to a fitted class variable $\hat{G}$ according to the rule

$$\hat{G} = \begin{cases} \texttt{RED} & \text{if } \hat{Y} > 0.5, \\ \texttt{GREEN} & \text{if } \hat{Y} \leq 0.5. \end{cases} \tag{2.7}$$

The set of points in $\mathbb{R}^2$ classified as RED corresponds to $\{x : x^T\hat{\beta} > 0.5\}$, indicated in Figure 2.1, and the two predicted classes are separated by the *decision boundary* $\{x : x^T\hat{\beta} = 0.5\}$, which is linear in this case. We see that for these data there are several misclassifications on both sides of the decision boundary. Perhaps our linear model is too rigid— or are such errors unavoidable? Remember that these are errors on the training data itself, and we have not said where the constructed data came from. Consider the two possible scenarios:

Scenario 1: The training data in each class were generated from bivariate Gaussian distributions with uncorrelated components and different means.

Scenario 2: The training data in each class came from a mixture of 10 low-variance Gaussian distributions, with individual means themselves distributed as Gaussian.

A mixture of Gaussians is best described in terms of the generative model. One first generates a discrete variable that determines which of the component Gaussians to use, and then generates an observation from the chosen density. In the case of one Gaussian per class, we will see in Chapter 4 that a linear decision boundary is the best one can do, and that our estimate is almost optimal. The region of overlap is inevitable, and future data to be predicted will be plagued by this overlap as well.

In the case of mixtures of tightly clustered Gaussians the story is different. A linear decision boundary is unlikely to be optimal, and in fact is not. The optimal decision boundary is nonlinear and disjoint, and as such will be much more difficult to obtain.

We now look at another classification and regression procedure that is in some sense at the opposite end of the spectrum to the linear model, and far better suited to the second scenario.

2.3.2 Nearest-Neighbor Methods

Nearest-neighbor methods use those observations in the training set $\mathcal{T}$ closest in input space to x to form $\hat{Y}$. Specifically, the k-nearest neighbor fit for $\hat{Y}$ is defined as follows:

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i, \tag{2.8}$$

where $N_k(x)$ is the neighborhood of x defined by the k closest points x_i in the training sample. Closeness implies a metric, which for the moment we assume is Euclidean distance. So, in words, we find the k observations with x_i closest to x in input space, and average their responses.

In Figure 2.2 we use the same training data as in Figure 2.1, and use 15-nearest-neighbor averaging of the binary coded response as the method of fitting. Thus $\hat{Y}$ is the proportion of RED's in the neighborhood, and so assigning class RED to $\hat{G}$ if $\hat{Y} > 0.5$ amounts to a majority vote in the neighborhood. The colored regions indicate all those points in input space classified as GREEN or RED by such a rule, in this case found by evaluating the procedure on a fine grid in input space. We see that the decision boundaries that separate the GREEN from the RED regions are far more irregular, and respond to local clusters where one class dominates.

Figure 2.3 shows the results for 1-nearest-neighbor classification: $\hat{Y}$ is assigned the value y_ℓ of the closest point x_ℓ to x in the training data. In this case the regions of classification can be computed relatively easily, and correspond to a *Voronoi tessellation* of the training data. Each point x_i

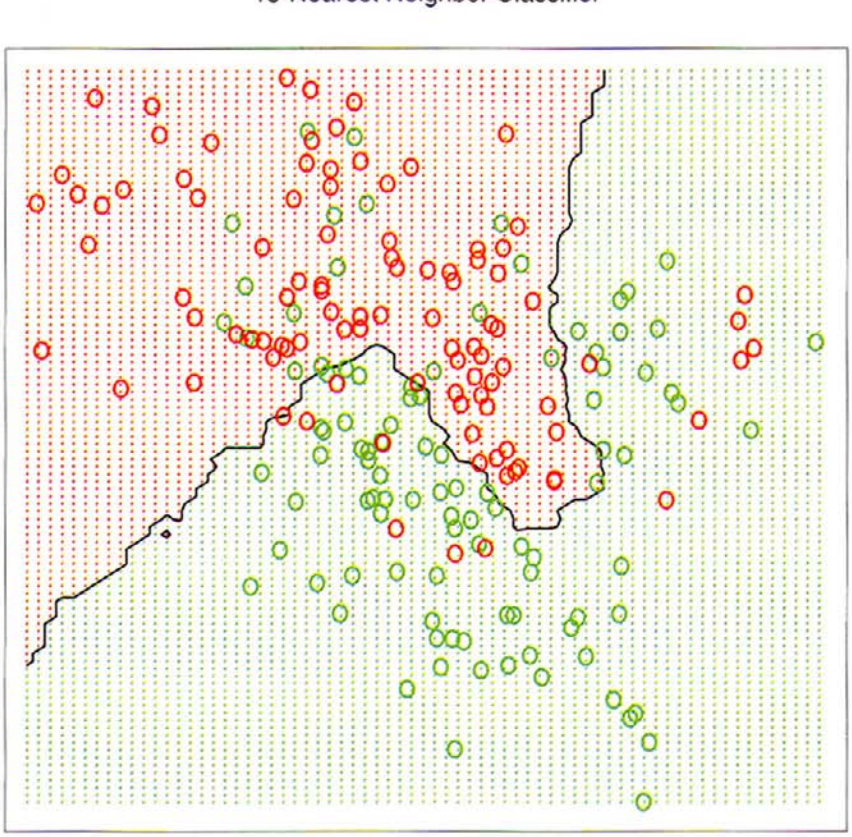

FIGURE 2.2. *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable* (GREEN $= 0$, RED $= 1$) *and then fit by* 15*-nearest-neighbor averaging as in (2.8). The predicted class is hence chosen by majority vote amongst the* 15*-nearest neighbors.*

has an associated tile bounding the region for which it is the closest input point. For all points x in the tile, $\hat{G}(x) = g_i$. The decision boundary is even more irregular than before.

The method of k-nearest-neighbor averaging is defined in exactly the same way for regression of a quantitative output Y, although $k = 1$ would be an unlikely choice.

In Figure 2.2 we see that far fewer training observations are misclassified than in Figure 2.1. This should not give us too much comfort, though, since in Figure 2.3 *none* of the training data are misclassified. A little thought suggests that for k-nearest-neighbor fits, the error on the training data should be approximately an increasing function of k, and will always be 0 for $k = 1$. An independent test set would give us a more satisfactory means for comparing the different methods.

It appears that k-nearest-neighbor fits have a single parameter, the number of neighbors k, compared to the p parameters in least-squares fits. Although this is the case, we will see that the *effective* number of parameters of k-nearest neighbors is N/k and is generally bigger than p, and decreases with increasing k. To get an idea of why, note that if the neighborhoods

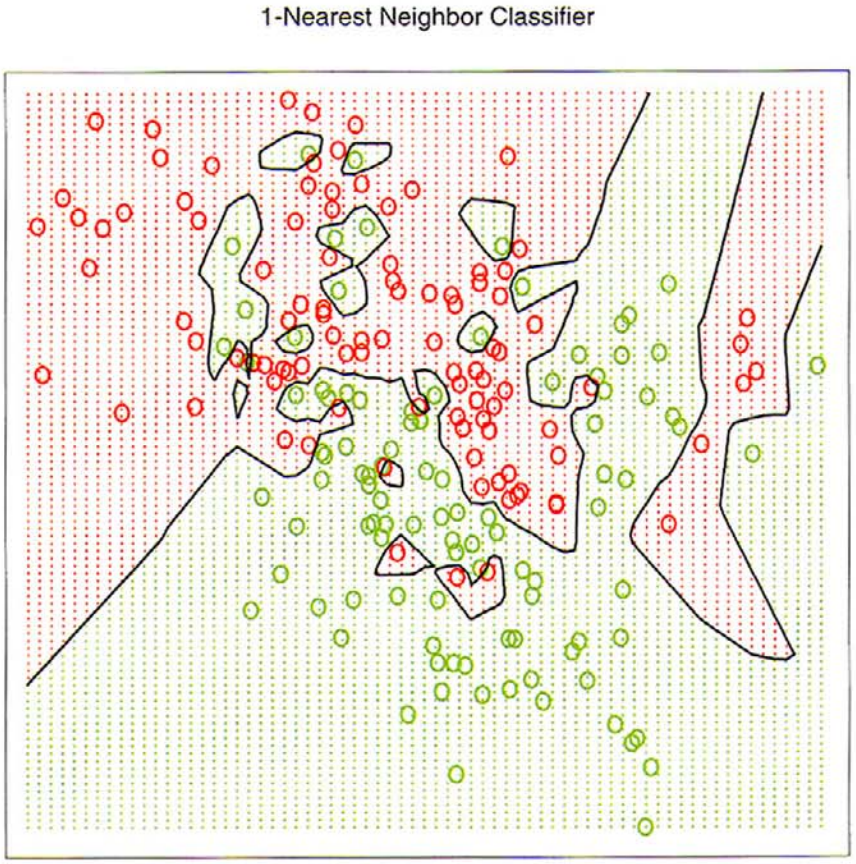

FIGURE 2.3. *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable* (GREEN $= 0$, RED $= 1$), *and then predicted by* 1*-nearest-neighbor classification.*

were nonoverlapping, there would be N/k neighborhoods and we would fit one parameter (a mean) in each neighborhood.

It is also clear that we cannot use sum-of-squared errors on the training set as a criterion for picking k, since we would always pick $k = 1$! It would seem that k-nearest-neighbor methods would be more appropriate for the mixture Scenario 2 described above, while for Gaussian data the decision boundaries of k-nearest neighbors would be unnecessarily noisy.

2.3.3 From Least Squares to Nearest Neighbors

The linear decision boundary from least squares is very smooth, and apparently stable to fit. It does appear to rely heavily on the assumption that a linear decision boundary is appropriate. In language we will develop shortly, it has low variance and potentially high bias.

On the other hand, the k-nearest-neighbor procedures do not appear to rely on any stringent assumptions about the underlying data, and can adapt to any situation. However, any particular subregion of the decision boundary depends on a handful of input points and their particular positions, and is thus wiggly and unstable—high variance and low bias.

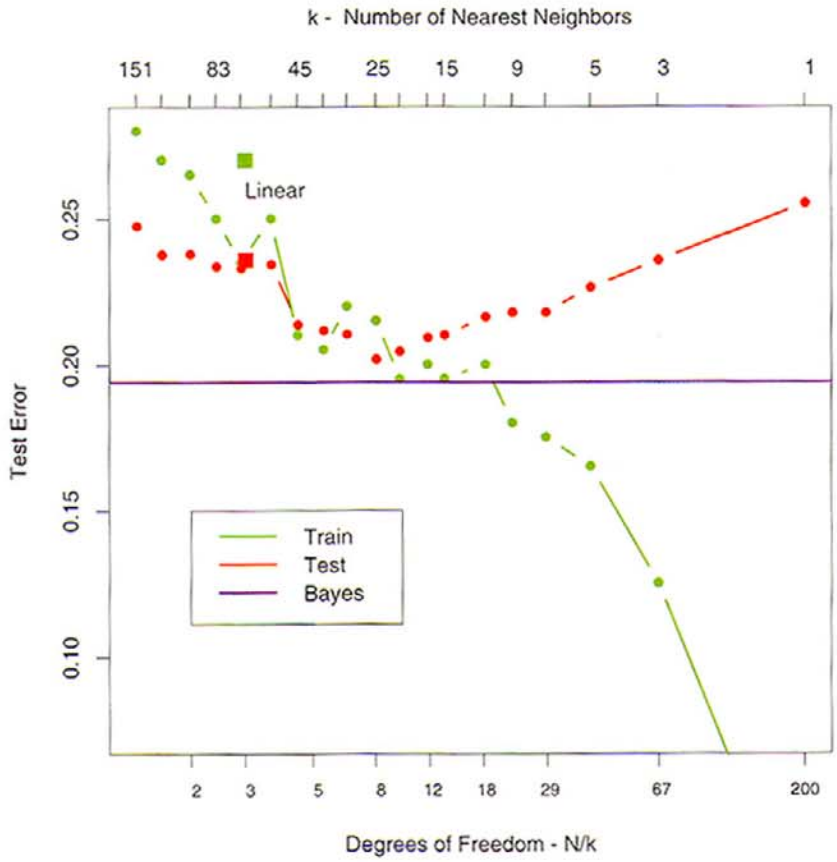

FIGURE 2.4. *Misclassification curves for the simulation example used in Figures 2.1, 2.2 and 2.3. A single training sample of size* 200 *was used, and a test sample of size* 10,000. *The red curves are test and the green are training error for k-nearest-neighbor classification. The results for linear regression are the bigger green and red dots at three degrees of freedom. The purple line is the optimal Bayes Error Rate.*

Each method has its own situations for which it works best; in particular linear regression is more appropriate for Scenario 1 above, while nearest neighbors are more suitable for Scenario 2. The time has come to expose the oracle! The data in fact were simulated from a model somewhere between the two, but closer to Scenario 2. First we generated 10 means m_k from a bivariate Gaussian distribution $N((1,0)^T, \mathbf{I})$ and labeled this class GREEN. Similarly, 10 more were drawn from $N((0,1)^T, \mathbf{I})$ and labeled class RED. Then for each class we generated 100 observations as follows: for each observation, we picked an m_k at random with probability 1/10, and then generated a $N(m_k, \mathbf{I}/5)$, thus leading to a mixture of Gaussian clusters for each class. Figure 2.4 shows the results of classifying 10,000 new observations generated from the model. We compare the results for least squares and those for k-nearest neighbors for a range of values of k.

A large subset of the most popular techniques in use today are variants of these two simple procedures. In fact 1-nearest-neighbor, the simplest of all, captures a large percentage of the market for low-dimensional problems.

The following list describes some ways in which these simple procedures have been enhanced:

- Kernel methods use weights that decrease smoothly to zero with distance from the target point, rather than the effective 0/1 weights used by k-nearest neighbors.
- In high-dimensional spaces the distance kernels are modified to emphasize some variable more than others.
- Local regression fits linear models by locally weighted least squares, rather than fitting constants locally.
- Linear models fit to a basis expansion of the original inputs allow arbitrarily complex models.
- Projection pursuit and neural network models consist of sums of nonlinearly transformed linear models.

2.4 Statistical Decision Theory

In this section we develop a small amount of theory that provides a framework for developing models such as those discussed informally so far. We first consider the case of a quantitative output, and place ourselves in the world of random variables and probability spaces. Let $X \in \mathbb{R}^p$ denote a real valued random input vector, and $Y \in \mathbb{R}$ a real valued random output variable, with joint distribution $\Pr(X, Y)$. We seek a function $f(X)$ for predicting Y given values of the input X. This theory requires a *loss function* $L(Y, f(X))$ for penalizing errors in prediction, and by far the most common and convenient is *squared error loss*: $L(Y, f(X)) = (Y - f(X))^2$. This leads us to a criterion for choosing f,

$$\begin{aligned} \mathrm{EPE}(f) &= \mathrm{E}(Y - f(X))^2 && (2.9) \\ &= \int (y - f(x))^2 \Pr(dx, dy), && (2.10) \end{aligned}$$

the expected (squared) prediction error. By conditioning* on X, we can write EPE as

$$\mathrm{EPE}(f) = \mathrm{E}_X \mathrm{E}_{Y|X} \left([Y - f(X)]^2 | X \right) \tag{2.11}$$

and we see that it suffices to minimize EPE pointwise:

$$f(x) = \mathrm{argmin}_c \mathrm{E}_{Y|X} \left([Y - c]^2 | X = x \right). \tag{2.12}$$

*Conditioning here amounts to factoring the joint density $\Pr(X, Y) = \Pr(Y|X)\Pr(X)$ where $\Pr(Y|X) = \Pr(Y, X)/\Pr(X)$, and splitting up the bivariate integral accordingly

The solution is

$$f(x) = \mathrm{E}(Y|X = x), \tag{2.13}$$

the conditional expectation, also known as the *regression* function. Thus the best prediction of Y at any point $X = x$ is the conditional mean, when best is measured by average squared error.

The nearest-neighbor methods attempt to directly implement this recipe using the training data. At each point x, we might ask for the average of all those y_is with input $x_i = x$. Since there are typically at most one observation at any point x, we settle for

$$\hat{f}(x) = \mathrm{Ave}(y_i|x_i \in N_k(x)), \tag{2.14}$$

where "Ave" denotes average, and $N_k(x)$ is the neighborhood containing the k points in T closest to x. Two approximations are happening here:

- expectation is approximated by averaging over sample data;
- conditioning at a point is relaxed to conditioning on some region "close" to the target point.

For large training sample size N, the points in the neighborhood are likely to be close to x, and as k gets large the average will get more stable. In fact, under mild regularity conditions on the joint probability distribution $\Pr(X, Y)$, one can show that as $N, k \to \infty$ such that $k/N \to 0$, $\hat{f}(x) \to \mathrm{E}(Y|X = x)$. In light of this, why look further, since it seems we have a universal approximator? We often do not have very large samples. If the linear or some more structured model is appropriate, then we can usually get a more stable estimate than k-nearest neighbors, although such knowledge has to be learned from the data as well. There are other problems though, sometimes disastrous. In Section 2.5 we see that as the dimension p gets large, so does the metric size of the k-nearest neighborhood. So settling for nearest neighborhood as a surrogate for conditioning will fail us miserably. The convergence above still holds, but the *rate* of convergence decreases as the dimension increases.

How does linear regression fit into this framework? The simplest explanation is that one assumes that the regression function $f(x)$ is approximately linear in its arguments:

$$f(x) \approx x^T \beta. \tag{2.15}$$

This is a model-based approach—we specify a model for the regression function. Plugging this linear model for $f(x)$ into EPE (2.9) and differentiating we can solve for β theoretically:

$$\beta = [\mathrm{E}(XX^T)]^{-1}\mathrm{E}(XY). \tag{2.16}$$

Note we have *not* conditioned on X; rather we have used our knowledge of the functional relationship to *pool* over values of X. The least squares solution (2.6) amounts to replacing the expectation in (2.16) by averages over the training data.

So both k-nearest neighbors and least squares end up approximating conditional expectations by averages. But they differ dramatically in terms of model assumptions:

- Least squares assumes $f(x)$ is well approximated by a globally linear function.
- k-nearest neighbors assumes $f(x)$ is well approximated by a locally constant function.

Although the latter seems more palatable, we have already seen that we may pay a price for this flexibility.

Many of the more modern techniques described in this book are model based, although far more flexible than the rigid linear model. For example, additive models assume that

$$f(X) = \sum_{j=1}^{p} f_j(X_j). \tag{2.17}$$

This retains the additivity of the linear model, but each coordinate function f_j is arbitrary. It turns out that the optimal estimate for the additive model uses techniques such as k-nearest neighbors to approximate *univariate* conditional expectations *simultaneously* for each of the coordinate functions. Thus the problems of estimating a conditional expectation in high dimensions are swept away in this case by imposing some (often unrealistic) model assumptions, in this case additivity.

Are we happy with the criterion (2.11)? What happens if we replace the L_2 loss function with the L_1: $E|Y - f(X)|$? The solution in this case is the conditional median,

$$\hat{f}(x) = \text{median}(Y|X = x), \tag{2.18}$$

which is a different measure of location, and its estimates are more robust than those for the conditional mean. L_1 criteria have discontinuities in their derivatives, which have hindered their widespread use. Other more resistant loss functions will be mentioned in later chapters, but squared error is analytically convenient and the most popular.

What do we do when the output is a categorical variable G? The same paradigm works here, except we need a different loss function for penalizing prediction errors. An estimate $\hat{G}$ will assume values in $\mathcal{G}$, the set of possible classes. Our loss function can be represented by a $K \times K$ matrix $\mathbf{L}$, where $K = \text{card}(\mathcal{G})$. $\mathbf{L}$ will be zero on the diagonal and nonnegative elsewhere,

where $L(k, \ell)$ is the price paid for classifying an observation belonging to class $\mathcal{G}_k$ as $\mathcal{G}_\ell$. Most often we use the *zero–one* loss function, where all misclassifications are charged a single unit. The expected prediction error is

$$\text{EPE} = \text{E}[L(G, \hat{G}(X))], \tag{2.19}$$

where again the expectation is taken with respect to the joint distribution $\Pr(G, X)$. Again we condition, and can write EPE as

$$\text{EPE} = \text{E}_X \sum_{k=1}^{K} L[\mathcal{G}_k, \hat{G}(X)]\Pr(\mathcal{G}_k|X) \tag{2.20}$$

and again it suffices to minimize EPE pointwise:

$$\hat{G}(x) = \text{argmin}_{g \in \mathcal{G}} \sum_{k=1}^{K} L(\mathcal{G}_k, g)\Pr(\mathcal{G}_k|X = x). \tag{2.21}$$

With the 0–1 loss function this simplifies to

$$\hat{G}(x) = \text{argmin}_{g \in \mathcal{G}}[1 - \Pr(g|X = x)] \tag{2.22}$$

or simply

$$\hat{G}(X) = \mathcal{G}_k \text{ if } \Pr(\mathcal{G}_k|X = x) = \max_{g \in \mathcal{G}} \Pr(g|X = x). \tag{2.23}$$

This reasonable solution is known as the *Bayes classifier*, and says that we classify to the most probable class, using the conditional (discrete) distribution $\Pr(G|X)$. Figure 2.5 shows the Bayes-optimal decision boundary for our simulation example. The error rate of the Bayes classifier is called the *Bayes rate*.

Again we see that the k-nearest neighbor classifier directly approximates this solution—a majority vote in a nearest neighborhood amounts to exactly this, except that conditional probability at a point is relaxed to conditional probability within a neighborhood of a point, and probabilities are estimated by training-sample proportions.

Suppose for a two-class problem we had taken the dummy-variable approach and coded G via a binary Y, followed by squared error loss estimation. Then $\hat{f}(X) = \text{E}(Y|X) = \Pr(G = \mathcal{G}_1|X)$ if $\mathcal{G}_1$ corresponded to $Y = 1$. Likewise for a K-class problem, $\text{E}(Y_k|X) = \Pr(G = \mathcal{G}_k|X)$. This shows that our dummy-variable regression procedure, followed by classification to the largest fitted value, is another way of representing the Bayes classifier. Although this theory is exact, in practice problems can occur, depending on the regression model used. For example, when linear regression is used, $\hat{f}(X)$ need not be positive, and we might be suspicious about using it as an estimate of a probability. We will discuss a variety of approaches to modeling $\Pr(G|X)$ in Chapter 4.

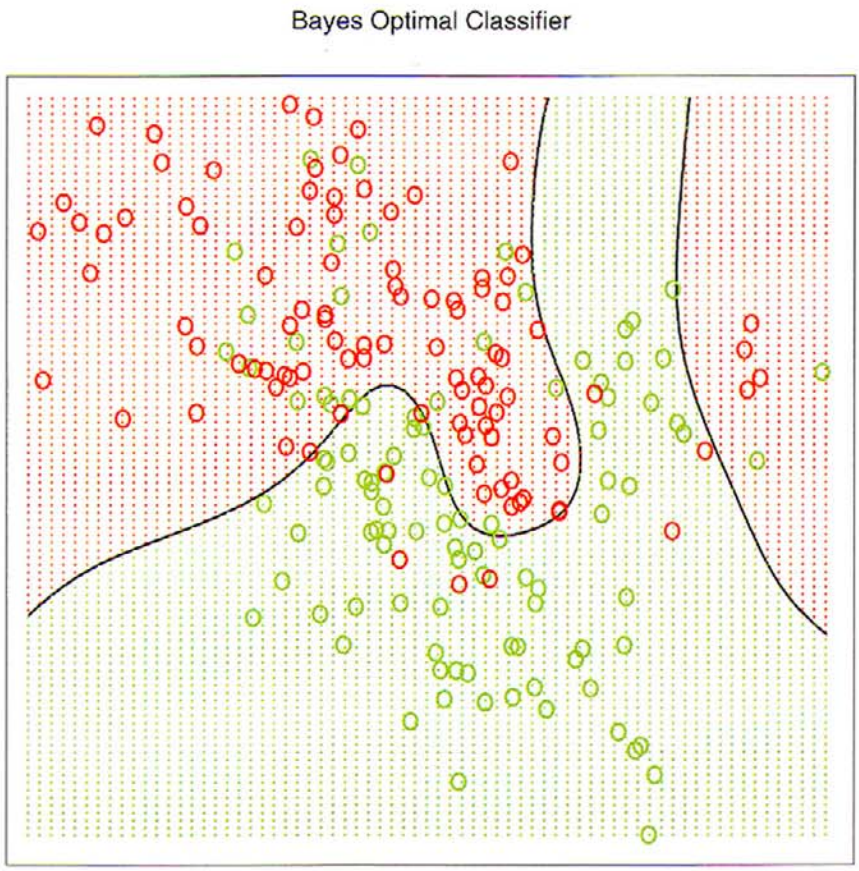

FIGURE 2.5. *The optimal Bayes decision boundary for the simulation example of Figures 2.1, 2.2 and 2.3. Since the generating density is known for each class, this boundary can be calculated exactly (Exercise 2.2).*

2.5 Local Methods in High Dimensions

We have examined two learning techniques for prediction so far: the stable but biased linear model and the less stable but apparently less biased class of k-nearest-neighbor estimates. It would seem that with a reasonably large set of training data, we could always approximate the theoretically optimal conditional expectation by k-nearest-neighbor averaging, since we should be able to find a fairly large neighborhood of observations close to any x and average them. This approach and our intuition breaks down in high dimensions, and the phenomenon is commonly referred to as the *curse of dimensionality* (Bellman, 1961). There are many manifestations of this problem, and we will examine a few here.

Consider the nearest-neighbor procedure for inputs uniformly distributed in a p-dimensional unit hypercube, as in Figure 2.6. Suppose we send out a hypercubical neighborhood about a target point to capture a fraction r of the observations. Since this corresponds to a fraction r of the unit volume, the expected edge length will be $e_p(r) = r^{1/p}$. In ten dimensions $e_{10}(0.01) = 0.63$ and $e_{10}(0.1) = 0.80$, while the entire range for each input is only 1.0.

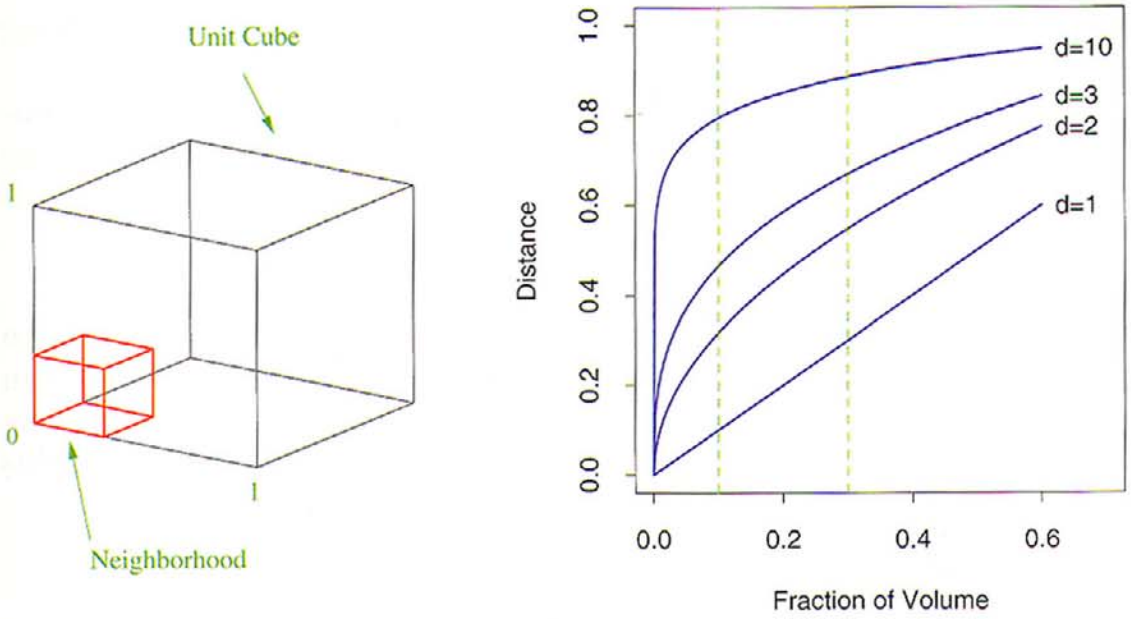

FIGURE 2.6. *The curse of dimensionality is well illustrated by a subcubical neighborhood for uniform data in a unit cube. The figure on the right shows the side-length of the subcube needed to capture a fraction r of the volume of the data, for different dimensions p. In ten dimensions we need to cover 80% of the range of each coordinate to capture 10% of the data.*

So to capture 1% or 10% of the data to form a local average, we must cover 63% or 80% of the range of each input variable. Such neighborhoods are no longer "local." Reducing r dramatically does not help much either, since the fewer observations we average, the higher is the variance of our fit.

Another consequence of the sparse sampling in high dimensions is that all sample points are close to an edge of the sample. Consider N data points uniformly distributed in a p-dimensional unit ball centered at the origin. Suppose we consider a nearest-neighbor estimate at the origin. The median distance from the origin to the closest data point is given by the expression

$$d(p, N) = \left(1 - \frac{1}{2}^{1/N}\right)^{1/p} \tag{2.24}$$

(Exercise 2.3). A more complicated expression exists for the mean distance to the closest point. For $N = 5000$, $p = 10$, $d(p, N) \approx 0.52$, more than half way to the boundary. Hence most data points are closer to the boundary of the sample space than to any other data point. The reason that this presents a problem is that prediction is much more difficult near the edges of the training sample. One must extrapolate from neighboring sample points rather than interpolate between them.

Another manifestation of the curse is that the sampling density is proportional to $N^{1/p}$, where p is the dimension of the input space and N is the sample size. Thus if $N_1 = 100$ represents a dense sample for a single input problem, then $N_{10} = 100^{10}$ is the sample size required for the same sam-