A model is a high-level, global description of a data set. It takes a large sample perspective. It may be descriptive—summarizing the data in a convenient and concise way—or it may be inferential, allowing one to make some statement about the population from which the data were drawn or about likely future data values. In this chapter we will discuss a variety of basic model forms such as linear regression models, mixture models, and Markov models.

In contrast, a pattern is a local feature of the data, perhaps holding for only a few records or a few variables (or both). An example of a pattern would be a local "structural" feature in our *p*-dimensional variable space such as a mode (or a gap) in a density function or an inflexion point in a regression curve. Often patterns are of interest because they represent departures from the general run of the data: a pair of variables that have a particularly high correlation, a set of items that have exceptionally high values on some variables, a group of records that always score the same on some variables, and so on. As with models, we may want to find patterns for descriptive reasons or for inferential reasons. We may want to identify members of the existing database that have unusual properties, or we may want to predict which future records are likely to have unusual properties. Examples of patterns are transient waveforms in an EEG trace, unusual combinations of products that are frequently purchased together by retail customers, and outliers in a database of semiconductor manufacturing data.

Data compression can provide a useful way to illustrate the concept of a patterns versus a model. Consider transmitter *T* that has an image *I* that is to be sent to a receiver *R* (though the principle holds for data sets that are not images). There are two main strategies: (a) send all of the data (the pixels in the image *I*) exactly, or (b) transmit some compressed version of the image#x2014;that is, some summary of the image *I*. Data mining to a large extent corresponds to the second approach, the compression being achieved either by representing the original data as a model, or by identifying unusual features of the data through patterns.

In modeling, some loss in fidelity is likely to be incurred when we summarize the data—this means that the receiver *R* will not be able to reconstruct the data precisely. An example of a model for the image data might be replacing each square of 16 × 16 pixels in the original image by the average values of these pixels. The "model" in this case would just be a set of smaller and lower resolution (1/16th) images. A more sophisticated model might adaptively partition each image into local regions of different sizes and shapes, where the pixel values can be fairly accurately described by a constant pixel intensity within each such region. The "model" (or message) in this case would be both the values of the constants within each region and the description of the boundaries of the regions for each. For both types of models (the average-pixel model and the locally constant model) it is clear that the complexity of the image model (the number of pixels being averaged, the average size of the locally constant regions) can be traded for the amount of information being transmitted (or equivalently, the amount of information being lost in the transmission—that is, the compression rate).

From a pattern detection viewpoint, a pattern in an image is some structure in the image that is purely local: for example, a partially obscured circular object in the upper-left corner of the image. This is clearly a different form of compression from the global compression models above. The receiver *R* can no longer reconstruct a summary of the *whole image*, but it does have a description of some *local part* of the image. Depending on the problem and objectives, local structure may be much more relevant than a global model. Rather than sending a summary model description of a vast noisy "sea" of pixel values, the transmitter *T* instead "focuses" the receiver *R*'s attention on the important aspects. We can think of association rules from chapter 5 in this context: they try to focus attention on potentially interesting associations among subsets of variables.

The analogy between image coding and data analysis is not perfect (for example, compression, as we have described it, does not take into account the idea of generalization to unseen data), but nonetheless, it allows us to grasp the essential trade-offs between representing local structure at a fairly high resolution and lower-resolution global structure.

This chapter is organized as follows: section 6.2 discusses some of the fundamental properties of models and the choices we have to make in building a model. Section 6.3 focuses on the general principles behind models in which one of the variables is singled

out as a "response" to be predicted from the others. This includes regression and supervised classification models. Many data mining problems involve large numbers of variables, and these present particular challenges, which are discussed in section 6.5. Descriptive models are discussed in section 6.4. Many data sets contain data that have been collected to conform to some schema (such as time series or image data), and they typically require special consideration in modeling. Section 6.6 discusses issues associated with such structured data. Finally, section 6.7 describes patterns for both multivariate and sequential data.

6.2 Fundamentals of Modeling

A model is an abstract representation of a real-world process. For example, $Y = 3X + 2$ is a very simple model of how the variable Y might relate to the variable X. This particular model can be thought of as an instance of the more general *model structure* $Y = aX + c$, where for this particular model we have set $a = 3$ and $c = 2$. More generally still, we could put $Y = aX + c + e$, where e is a random variable accounting for a random component of the mapping from X to Y (we will return to this later). We often refer to a and c as the *parameters* of the model, and will often use the notation ? to refer to a generic parameter or a set (or vector) of parameters, as we did in chapter 4. In this example, ? = {*a, c*}. Given the form or structure of a model, we choose appropriate values for its parameters by estimation—that is, by minimizing or maximizing an appropriate score function measuring the fit of the model to the data. Procedures for this were described in chapter 4 and are described further in later chapters.

However, before we can estimate the parameters of a model, we must first choose an appropriate functional form of the model itself. The aim of this section is to present a high-level overview of the main classes of models used in data mining.

Model building in data mining is data-driven. It is usually not driven by the notion of any underlying mechanism or "reality," but simply seeks to capture the relationships in the data. Even in those cases in which there is a postulated true generative mechanism for the data, we should bear in mind that, as George Box put it, "All models are wrong but some are useful." For example, while we might postulate the existence of a linear model to explain the data, it is likely to be a fiction, since even in the best of circumstances there will be small nonlinear effects that we will be unable to capture in the model. We are looking for a model that encapsulates the main aspects of the data generating process.
Since data mining is data-driven, the discovery of a highly predictive model (for example) should not be taken to mean that there is a causal relationship. For example, an analysis of customer records may show that customers who buy high-quality wines are also more likely to buy designer clothes. Clearly one propensity is not causally related to the other propensity (in either direction). Rather, they are both more likely to be the consequence of a relatively high income. However, the fact that neither the wine nor the clothes variable causes the other does not mean that they are not useful for predictive purposes. Predicting the likely clothes-buying behavior from observed wine-buying behavior would be entirely legitimate (if the relationship were found in the data), from a marketing perspective. Since no causal relationship has been established, however, it would be false to conclude that *manipulating* one of the variables would lead to a change in the other. That is, inducing people to buy high-quality wines would be unlikely to lead them also to buy designer clothes, even if the relationship existed in the data.

6.3 Model Structures for Prediction

In a predictive model, one of the variables is expressed as a function of the others. This permits the value of the *response* variable to be predicted from given values of the others (the *explanatory* or *predictor* variables). The response variable in general predictive models is often denoted by Y , and the p predictor variables by $X_1, \ldots, X_p$. Thus, for example, we might want to construct a model for predicting the probability that an applicant for a loan will default, based on application forms and the behavior of past

customers contained in a database. The record for the *i*th past customer can be conveniently represented as $\{(\mathbf{x}(i), y(i))\}$. Here $y(i)$ is the outcome class (good or bad) of the *i*th customer, and $\mathbf{x}(i)$ is the vector $\mathbf{x} = (x_1(i), \ldots, x_p(i))$ of application form values for the *i*th customer. The model will yield predictions, $y = f(x_1, \ldots, x_p; ?)$ where y is the prediction of the model and ? represents the parameters of the model structure. When *Y* is quantitative, this task of estimating a mapping from the *p*-dimensional **X** to *Y* is known as *regression*. When *Y* is categorical, the task of learning a mapping from **X** to *Y* is called *classification learning* or *supervised classification*. Both of these tasks can be considered *function approximation* problems in that we are learning a mapping from a *p*-dimensional variable **X** to *Y*. For simplicity of exposition in this chapter we will focus primarily on the regression task, since many of the same general principles carry over directly to the classification task. Chapters 10 and 11 deal, respectively, with supervised classification and regression in detail.

6.3.1 Regression Models with Linear Structure

We begin our discussion of predictive models with models in which the response variable is a linear function of the predictor variables:

(6.1) $$\hat{Y} = a_0 + \sum_{j=1}^{p} a_j X_j$$

where $? = \{a_0, \ldots, a_p\}$. Again we note that the model is purely empirical, so that the existence of a well-fitting and highly predictive model does not imply any causal relationship. We have used Y rather than simply *Y* on the left of this expression because it is a *model*, which has been constructed from the data. That is, the values of Y are values predicted from the **X**, and not values actually observed. This distinction is discussed in more detail in chapter 11.

Geometrically, this model describes a *p*-dimensional hyperplane embedded in a $(p + 1)$-dimensional space with slope determined by the a_j's and intercept by a_0. The aim of parameter estimation is to choose the *a* values to locate and angle this hyperplane so as to provide the best fit to the data $\{(\mathbf{x}(i), y(i))\}$, $i = 1, \ldots, n$, where the quality of fit is measured in terms of the differences between observed *y* values and the values y predicted from the model.

Models with this type of linear structure hold a special place in the history of data analysis, partly because estimation of parameters is straightforward with appropriate score functions, and partly because the structure of the model is simple and easy to interpret. For example, the *additive* nature of the model means that the parameters tell us the effect of changing any one of the predictor variables "keeping the others constant." Of course, there are circumstances in which the notion of *individual contribution* makes little sense. In particular, if two variables are highly correlated, then it is not meaningful to talk of the contribution from changing one while "holding the other constant." Such issues are discussed in more detail in later chapters.

We can retain the additive nature of the model, while generalizing beyond linear functions of the predictor variables. Thus

(6.2) $$\hat{Y} = a_0 + \sum_{j=1}^{p} a_j f_j(X_j)$$

where the f_j functions are smooth (but possibly nonlinear) functions of the X_js. For example, the f_js could be log, square-root, or related transformations of the original *X* variables. This model still assumes that the dependent variable *Y* depends on the independent variables in the model (the *X*s) in an additive fashion. Again, this may be a strong assumption in practice, but it will lead to a model in which it may be easy to interpret the contribution of each individual *X* variable. The simplicity of the model also means that there are relatively few parameters $(p + 1)$ to estimate from the data, making the estimation problem relatively straightforward.

We can also generalize this linear model structure to allow general polynomials in the *X*s with cross-product terms to allow interaction among the X_js in the model. The one-dimensional case is again familiar—we can imagine a 2nd or 3rd or *k*th order polynomial interpolating the observed *y* values. The multidimensional case generalizes this so that we have a smooth surface defined on *p* variables in the $(p + 1)$-dimensional space.

Note in passing that even though these predictive models are *nonlinear* in the variables X, they are still *linear* in the parameters. This makes estimation of these parameters much easier than in the case where the parameters enter in a nonlinear fashion, as we will see in chapter 11.

Example 6.1

In figure 6.1(a) we show a set of 50 data points that are simulated from the equation $y = 0.001x^3 - 0.05x^2 + x^3 + e$, where e is additive Gaussian noise (zero mean, standard deviation s = 3), over the range $x \in [1, 50]$. A linear fit to the data is shown in figure 6.1(b) and a second-order polynomial fit is shown in figure 6.1(c). Although the linear fit captures the general upward trend in Y as a function of X (over this particular range), the second-order fit is clearly better. Neither fit fully captures the underlying curvature of the true structure, as can be seen from the structure in the errors for each model (that is, the errors for each model have systematic structure as a function of x). Both fits were determined by minimizing a sum of squares score function.

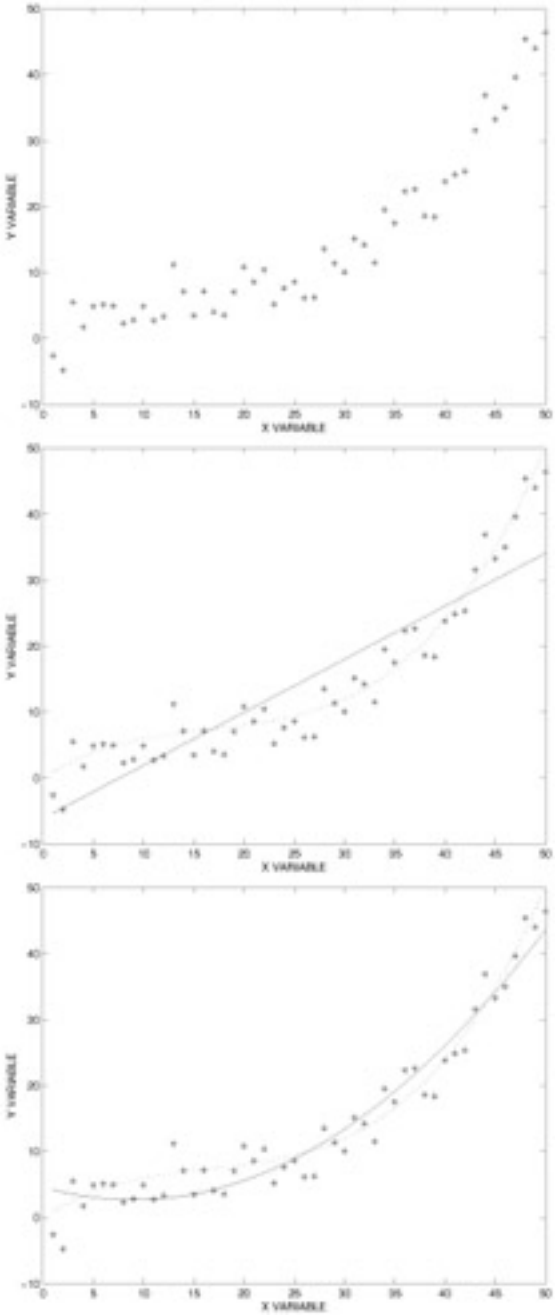

Figure 6.1: (a) Fifty Data Points that are Simulated From a Third-Order Polynomial Equation

with Additive Gaussian (Normal) Noise, (b) The Fit of the Model $aX + b$ (Solid Line), (c) The Fit of the Model $aX^2 + bX + c$ (Solid Line). The Dotted Lines in (b) and (c) Indicate the True Model From Which the Data Points were Generated (See Text). The Model Parameters in Each Case were Estimated by Minimizing the Sum of Squared Errors between Model Predictions and Observed Data.

Note that by allowing models with higher order terms and interactions between the components of **X** we can in principle estimate a more complex surface than with a simple linear model (a hyperplane). However, note that as *p* (the dimensionality of the input space) increases, the number of possible interaction terms in our model (such as X_jX_k) increases as a combinatorial function of *p*. Since each term has a weight coefficient (a parameter) in the additive model, the number of parameters to be estimated for the full model (with all possible interaction terms of order *k* among *p* variables) increases dramatically as *p* increases. The interpretation and understanding of such a model makes the estimation problem more difficult, and it also becomes increasingly difficult as *p* increases. A practical alternative is to select some small subset of the overall set of possible interactions to participate in the model. However, if the selection is carried out in a data-driven fashion (as is typically the fashion in a data mining application), the number of all possible interaction terms (the size of the search space) scales as 2^p, making the search problem exponentially more difficult as dimensionality *p* increases. We will return to this issue of how to handle dimensionality later in this chapter.

The generalization to polynomials brings up an important point, namely the *complexity* of the model. The more complex models contain the simpler models as special cases (so-called *nesting*). For example, the first-order $a_1X_1 + a_0$ model can be viewed as a special case of the 2nd order polynomial model $a_2X_1^2 + a_1X_1 + a_0$ by setting a_2 to zero. Thus, it is clear that a complex model (a high-order polynomial in the **X** variables) can always fit the observed data at least as well any simpler model can (since it includes any simpler model as a special case). In turn, this raises the complicated issue of how we should choose one model over another when the complexity (or expressive power) of each is different. This is a subtle question: we may want the model that is closest to some hypothesized unknown "truth"; we may want to find a model that captures the main features of the data without being too complicated; we may want to find the model that has the best predictive performance on data that it has not seen; and so on. We will return to this in later chapters. For now, however, we return to focus on the expressive capabilities of the models themselves without thinking yet of how we will choose among such models given observed data.

Transforming the predictor variables is one way to generalize a linear structure. Another way is to transform the response variable. *sqrt*(*Y*) may be perfectly related to a linear combination of the **X** variables, so that rather than fitting *Y* directly we may want to transform it by taking the square root first, and then use a linear combination of the **X** variables to predict *sqrt*(*Y*). Of course, we will not know beforehand that the square root is an appropriate transformation. We have to experiment, trying different transformations (and bearing in mind the constraints implied by the nature of the measurements involved, as discussed in chapter 2). This is why data mining is an exciting voyage of discovery, and not a mere exercise in applying standard tools in standard ways.

As we show in chapter 11, the simple linear regression model can be thought of as seeking to predict the expected value of the *Y* distribution at each value of the **X** predictors, namely $E[Y|X]$. That is, the regression model provides a prediction of a *parameter* of the conditional distribution of *Y* , where the parameter is the mean. More generally, of course, we can seek to predict other parameters of the conditional *Y* distribution from a linear combination of the **X** variables. This leads to the ideas of *generalized linear models* and *neural networks*, discussed in chapter 11.

We see that, although linear models are simple and easy to interpret (and, we will also see, their parameters can be easily estimated), they permit ready generalization to very powerful and flexible families of models. Any idea that the word *linear* implies a narrow class of models is illusory.

6.3.2 Local Piecewise Model Structures for Regression

Yet further generalizations of the basic linear model can be achieved if we assume that *Y* is *locally* linear in the *X*'s, with a different local dependence in various regions of the **X** space—that is, a piecewise linear model. Geometrically, our model structure consists of a *set* of different *p*-dimensional hyperplanes, each covering a region of the input (**X**) space disjoint from the others. The parameters of this model structure include both the local parameters for each hyperplane as well as the locations (boundaries) of the hyperplanes. For a one-dimensional *X* the picture is quite easy to visualize: a curve is approximated by *k* different line segments (see figure 6.2 for an example). Note that, in this figure, the line is continuous, with the line segments joining up. We could define a model structure that relaxes this, not requiring continuity at the ends of the line segments. This can be a useful model form, but sometimes the discontinuities can be problematic and undesirable because they imply a sudden jump in the predicted value of the response variable for an infinitesimal change in a predictor variable. To take an example, if a split between two line segments occurs at the value $50,000 for the variable income, we might get widely varying y predictions of the response variable, probability of loan default, for two applicants who are identical except that one earns $50,001 and the other earns $49,999. If the discontinuities are regarded as undesirable, one can go further and enforce continuity of derivatives of various orders at the end of the segments (which would clearly no longer be straight lines). Such curve segments are termed *splines*, with the whole model being a *spline function*. Typically, each line segment is taken to be a low-degree (quadratic or cubic) polynomial. The result is a smooth curve, but one that may change direction many times—the model would be highly flexible.

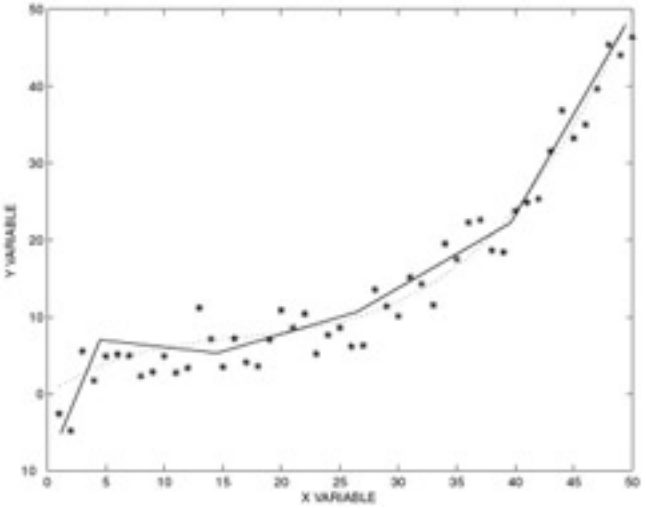

Figure 6.2: An Example of a Piecewise Linear Fit to the Data of Figure 6.1 with *k* = 5 Linear Segments.

These ideas can be generalized to more than one predictor variable. Again the local segments (which will now be (hyper)surfaces, not merely lines) may, but need not, join at their edges. Tree structures (described for supervised classification problems in chapter 10) provide an example of models of this form.

The piecewise linear model is a good example of how we can build relatively complex models for nonlinear phenomena by piecing together simple components (in this case hyperplanes). This is a recurring theme in data mining, the idea of composing complex global structures from relatively simple local components—and it also provides a link between ideas of modeling and ideas of pattern detection. That is, the locality also provides a framework for decomposing a complex model into simpler local patterns. For example, a "peak" in Y as a function of *X* will be reflected by two appropriately sloped line segments that adjoin each other.

This subsection and the preceding one together serve to show how complex models are built up from simpler ones, either by combining the simpler ones into more complex ones, or by generalizing them in various ways. No model used in data mining exists in splendid isolation. Rather, all such models are linked by a variety of connections, each being generalizations, special cases, or variants of others. The trick in effective model

building in data mining is to choose a model form that is well suited to answer the question being posed. This is not simply an exercise in choosing one model form, applying it, and presenting the conclusion. Rather, we fit a model, modify it or extend it in the light of the results, and repeat the exercise. Data mining, like data analysis in general, is an iterative process.

6.3.3 Nonparametric "Memory-Based" Local Models

In the preceding subsection we gave some examples of how models that are based on local characteristics of the data are related to, indeed are on a continuum including, broad global models. In this subsection we develop the ideas of local modeling further. (We recall that patterns, while also local, are isolated structures, and are not components of a global summary of the data. Thus we can talk of local modeling techniques as distinct from patterns.)

Roughly speaking, the spline and tree models briefly described above replace the data points by a function estimated from a neighborhood of data points. An alternative strategy is to retain the data points, and to leave the estimation of the predicted value of Y until the time at which a prediction is actually required. No longer are the data replaced by a function and its estimated parameters. For example, to estimate the value of a response variable Y for a new case, we could take the average of the Y values of the most similar k objects in the data set, where most similar is defined in terms of the predictor variables.

This idea has been extended to include all of the data set objects, but to weight them according to how similar they are to the new object—dissimilar ones will have small weight, similar ones large weight. The weight determines just how much their Y value contributes to the final estimate. An example of such an estimator is the *locally weighted regression* or *loess regression* model.

Although we have described the local smoothing ideas in a predictive modeling context, they can also be applied in a descriptive and density estimation context—which is the domain for which they were in fact first developed. Indeed, we have already seen an example of such methods for graphical display of a single variable in chapter 3 (where we used the ideas to estimate a probability density function), and we shall see more of them in later chapters. In this context, the so-called *kernel* estimators introduced in chapter 3 are common.

The obvious question with such estimators is how to determine the form of the weight function. A weight function that decays only slowly with decreasing similarity will lead to a smooth estimate, while one that decays rapidly will lead to a jagged estimate. A compromise must be found that is best suited to the aims of the analysis.

The weight function can be decomposed into two parts. One is its precise functional form, and the other is its "bandwidth." Thus, suppose that $K\left(\frac{x-z}{h}\right)$ is a smoothing function, which determines the contribution to the estimate at a new point z from a data set point at x. The size of this contribution will depend on the form of K and also on the size of the bandwidth h. A larger bandwidth h leads to a smoother function estimate, and a smaller bandwidth leads to a rougher, more jagged estimate. In practice, the precise form of the weight function turns out to be less important than the "band-width."

Example 6.2

Figure 6.3 shows an example of a regression function constructed with a triangular kernel using three different bandwidths. Here we are estimating the proportion of Nitrous oxide (NOx) in emissions as a function of ethanol (E), based on measurements taken on 81 automotive engines under different conditions. The widest bandwidth (h = 0.5) is clearly too broad, leading to an oversmoothed estimate that "misses" the central peak and the two tails. The narrowest bandwidth (h = 0.02) yields a very "spiky" estimate that appears to follow the noise in the observed data. The intermediate-valued bandwidth (h = 0.1) represents a reasonable trade-off, where the major features of the relationship between NOx and E are retained without overfitting. Subjective visual inspection can be useful technique for choosing bandwidths for simple one-dimensional problems, but does not

generalize well to the multidimensional case. One can also use more automated methods such as cross-validation for choosing *h* in a data-driven manner.

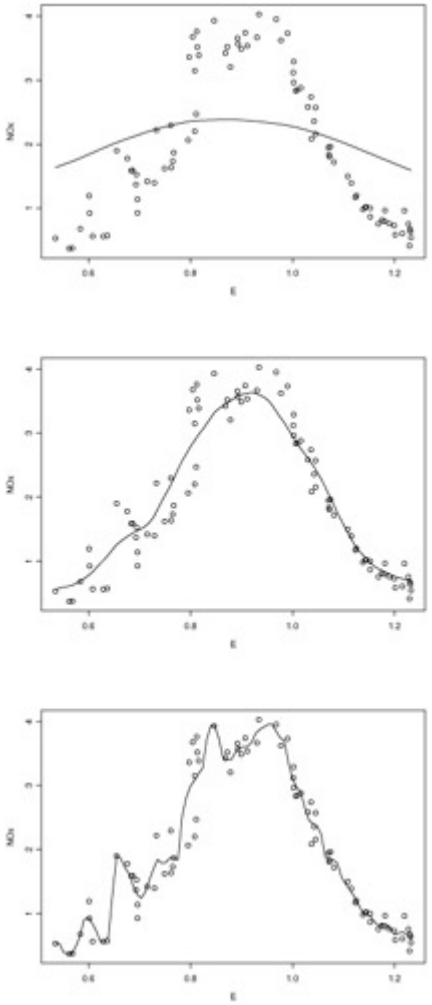

Figure 6.3: Nitrous Oxide (NOx) as a Function of Ethanol (E) using Kernel Regression with Triangular Kernels, With Bandwidths *h* = 0.5, 0.1, and 0.02, in Clockwise Order.

Kernel methods are closely related to *nearest neighbor methods*. Indeed, both classes of methods have now been extended and developed so that in some cases they are identical. Whereas kernel methods define the degree of smoothing in terms of a kernel function and bandwidth, nearest neighbor methods let the data determine the bandwidth by defining it in terms of the *number* of nearest neighbors. For example, the basic single nearest neighbor classifier (where *Y* is a class identifier) assigns a new object to the same class as its most similar object in the data set, and the *k*-nearest neighbor classifier assigns a new object to the most common class amongst the most similar *k* objects in the data set. More sophisticated nearest neighbor methods weight the contribution of according to distance from the point to be classified, and more sophisticated kernel methods let the bandwidth *h* depend on the data—so that they can be seen to be almost identical in terms of model structure.

Local model structures such as kernel models are often described as *non-parametric* because the model is largely data-driven with no parameters in the conventional sense (except for the bandwidth *h*). Such data-driven smoothing techniques (such as the kernel models) are useful for data interpretation, at least in one or two dimensions.

It will be clear that local models have their attractions. However, no model provides an answer to all problems, and local models have weaknesses. In particular, as the number of variables in the predictor space increases, so the number of data points required to

obtain accurate estimates increases exponentially (a consequence of the "curse of dimensionality"—see section 6.5 below). This means that these "local neighborhood" models tend to scale poorly to high dimensions.

Another drawback, particularly from a data mining viewpoint, is the lack of interpretability of the model. In low dimensions (p = 3 or so), we can plot the estimates. In high dimensions this is not possible, and there is no direct manner by which to summarize the model. Indeed, it is stretching the definition of a model to even call these representations models at all, since they are never explicitly defined as functions but instead are implicitly defined by the data.

6.3.4 Stochastic Components of Model Structures

Until this point, apart from a few brief references, we have ignored the fact that, with real data, we generally cannot find a perfect functional relationship between the predictor variables **X** and the response variable Y . That is, for any given vector of predictor variables **x**, more than one value of Y can be observed. The distribution of the values y at each value of **X** represents an aspect of variation that cannot be reduced by more sophisticated model building using just the variables in **X**. For this reason it is sometimes termed the *unexplainable* or *nonsystematic* or *random* component of the variation, with the variation in Y that can be explained in terms of the **X** variables being termed the *explainable* or *systematic* variation. (Of course merely because the systematic variation can be explained in principle by the variables in **X**, does not mean that we can necessarily build a model that will be able to do it).

In most of our discussion we have focused on the systematic component of the models, but we also need to consider the random component. The random component of models can arise from many sources. It can arise from simple measurement error—repeated measurements of Y will give different results, as discussed in chapter 2. The random component can also arise because our set of **X** variables does not include all of the variables that are required to make a perfect prediction of Y (for example, predicting whether a customer will purchase a particular product or not based only on past purchasing behavior will ignore potentially relevant demographic information about them such as age, income, and so on). Indeed, we should expect this usually to be the case—it would be a rare situation in which *all* of the variability in a variable was perfectly explained by just a handful of other variables, down to the finest detail.

Example 6.3

[redacted]

We can extend the regression modeling framework discussed earlier to include a stochastic component. We will assume that for any **x** we will observe a particular y but with some noise added; that is, there is some inherent uncertainty in the relationship between **x** and y:

(6.3) $y = g(\mathbf{x};\theta) + e$

where g(**x**; ?)) is a deterministic function of the inputs **x**, and e is often assumed to be a random variable (which is independent of **x**) with constant variance s^2 and zero-mean. The random term e can reflect noise in the measurement process (that is, we don't observe the "true" value for y but instead get a measurement of y which has random noise added). More generally, the random component e can reflect the fact that there are hidden variables (that are not being measured or are "hidden" from observation) that affect y in a manner that cannot be accounted for by the dependence of Y on the variables **X** alone.

The zero-mean assumption on e is fairly harmless, since if the noise has a constant non-zero mean it can be absorbed into g without loss of generality. If, for example, we make the common assumption that e has a Normal distribution with zero mean and constant variance s^2, then

(6.4) $y|\mathbf{x} \sim N(\mu_{y|\mathbf{x}}, \sigma^2), \quad \mu_{y|\mathbf{x}} = E[y|\mathbf{x}] = g(\mathbf{x};\theta).$

The constant s^2 assumption may require closer scrutiny in practice: for example, if Y represents the variable annual credit card spending, and X is income, it is plausible that the *variability* in Y will grow as a function of X. If this were the case, then to model this effect, s would need to be a function of x in the model above.

Note that the functional form of g is left free in these equations; that is, it could be chosen to be any of the various model structures we discussed earlier. We have already seen in chapter 4 that the Normal assumption on e above leads naturally to the principle of least-

squares regression—that is, finding the parameters ? that determine g such that $g(\mathbf{x}; ?)$ minimizes the sum of squares of between $f(\mathbf{x}; ?)$ and the observed y values.

The random component is important when it comes to choosing suitable score functions for estimating parameters and choosing between models. The likelihood score function, for example, introduced in chapter 4 and also discussed elsewhere, is based on assumptions about the form of the distribution of the random component. Extensions of the likelihood function that include a smoothness penalty so that too complex a model is not fitted also require assumptions about the distribution of the random component. More advanced methods based on likelihood concepts (for example, so-called *quasi-likelihood methods*) relax detailed distributional assumptions, but still base their choice of parameter estimates on aspects of the distribution of the random component.

6.3.5 Predictive Models for Classification

So far we have concentrated on predictive models in which the variable to be predicted, Y, was quantitative. We now briefly consider the case of a categorical variable Y, taking only a few possible categorical values. This is a *(supervised) classification* problem, with the aim being to assign a new object to its correct class (that is, the correct Y category) on the basis of its observed $\mathbf{X}$ values.

In classification we are essentially interested in modeling the boundaries between classes. As with regression, we can could make simple parametric assumptions about the functional form of the boundaries. For example, a classic approach is to use a linear hyperplane in the p-dimensional $\mathbf{X}$ space to define a decision boundary between two classes. That is, the model partitions the $\mathbf{X}$-space into disjoint decision regions (one for each class), where the decision regions are separated by linear boundaries (see figure 6.4 for an example). A more complex model might allow higher-order polynomial terms, yielding smooth polynomial decision boundaries. If we allow very flexible non-linear forms for our boundaries we arrive at models such as the neural network classifiers discussed in chapter 5.

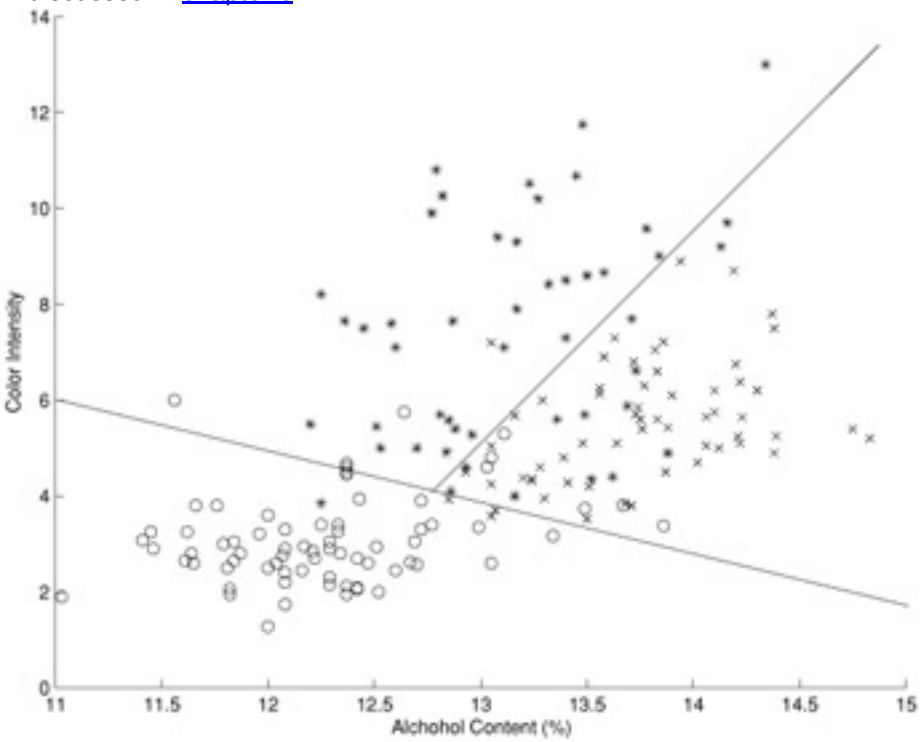

Figure 6.4: An Example of Linear Decision Boundaries for the Two-Dimensional Wine Classification Data Set of Chapter 5 (See Figure 5.1).

Just as in regression modeling, another way to allow more flexibility is to combine multiple simple local models, e.g., combinations of piecewise linear decision boundaries,

as in figure 6.5. For example, the classification tree models of chapter 5 define a particular class of local linear decision boundaries that are hierarchical and axis-parallel in structure. As mentioned earlier, the nearest-neighbor classifier is one where the class label of the nearest-neighbor from the training data set of a new unclassified data point is used for prediction. Although this technique is generally thought of as a method rather than a model per se, it does in fact implicitly define a piecewise linear decision boundary (at least when using Euclidean distance to define neighbors).

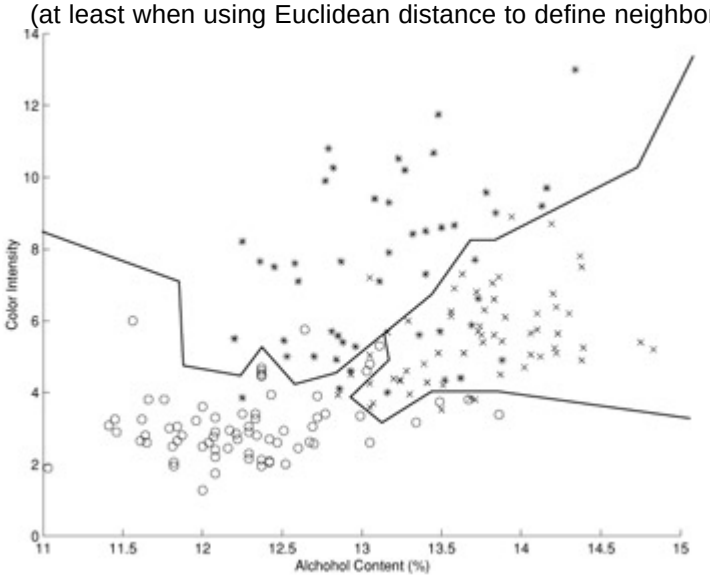

Figure 6.5: An Example of *Piecewise Linear* Decision Boundaries for the Two-Dimensional Wine Classification Data Set of Chapter 5 (See Figure 5.1).

There are a large number of different classification techniques, providing different ways to model decision boundaries. Something like nearest-neighbor is very flexible (allowing multiple local disjoint decision regions for each class, with flexible boundaries) whereas a single global hyperplane is a much simpler model.

From a practical modeling standpoint, prior knowledge about the shape of classification boundaries may not be as readily available as knowledge we may have about how Y is related to $\mathbf{X}$ in a regression problem. Nonetheless, the functional forms used successfully for discrimination models are quite similar to those we discussed earlier for regression modeling, and the same general themes emerge. We will return to classification models in much more detail in chapter 10 on classification.

6.3.6 An Aside: Selecting a Model of Appropriate Complexity

In our discussion so far we have seen that model structures range from the relatively simple to the complex. For example, in regression we saw that the complexity of a "piecewise-local" model structure is controlled by the number k of local regions (assuming that the complexity of the local function in each region is fixed). As we make k larger, we can obtain a curve that "follows" the observed data more closely. Put another way, the *expressive power* of the model structure increases in that it can represent more complex functions.

As we increase the expressive power of a model it is clear that we can in general continue to get a better fit to the available data. However, we need to be careful. While our score function on the training data may be improving, our model may actually be getting worse in terms of generalizing to new data. (Recall our discussion of this "overfitting" phenomenon in the context of classification trees in chapter 5, and figure 5.4 in particular). On the other hand, if we go the other direction and over-simplify our model structure, it may end being too simple. This issue of selecting a model of the appropriate complexity is always a key concern in any data analysis venture where we consider models of different complexities. In fact we will look at this from a theoretical viewpoint in chapter 7, using a generalization of the bias-variance trade-off that we first introduced in chapter 4.

In practice how can we choose a suitable compromise between simplicity and complexity? From a data-driven viewpoint (i.e., data mining) we can define a score

function that tries to estimate how well a model will perform on new data and not just on the training data. A commonly used approach is to combine both the usual goodness-of-fit term (on the training data) with an explicit second term to penalize model complexity. Another widely used approach is to partition the training data into two or more subsets (e.g., via cross-validation as described in chapter 5 for trees) and to train models on one subset and select models using a different validation data set.
Since the focus of this chapter is on the representational capabilities of different model and pattern structures (rather than on how they are scored relative to the data), we defer detail discussion of score functions to chapter 7. However, for the reader who up to this point was wondering how we would be able to select among the many different models being discussed here, the answer is that there do indeed exist well-defined data-driven score functions that allow us to search over different model structures in a principled manner to find what appears to be the best model for a given task (with some caveats as we will see in chapter 7).

6.4 Models for Probability Distributions and Density Functions

The previous section provided an overview of predictive problems, in which one of the variables (we labeled it *Y*) was singled out as special, to be predicted from the others. Many modeling problems in data mining fall into this class. However, many others are "descriptive," with the aim being simply to produce a summary or description of the data. If the available data are the complete data (for example, all chemical compounds of a certain type), then no notion of inference is relevant, and the aim is merely simplifying description. On the other hand, if the available data are a sample, or have been measured with error (so that collecting them again could yield slightly different values), then the aim is really one of inference—inferring the "true," or at least a good, model structure. In this latter case it is useful to think of the data as having been produced from an underlying probability function.

6.4.1 General Concepts

In this section we focus on some of the general classes of models used for density estimation (a more detailed discussion is given in chapter 9). While the functional form of the underlying models tend to be somewhat different from those we have seen earlier (for example, unimodal "bump" functions versus the linear and polynomial functions we saw for regression), several of the main concepts such as linear combinations of simpler models are once again widely applicable.

There are two general classes of distribution and density models:

1. **Parametric Models:** where a particular functional form is assumed. For real-valued variables the function is often characterized by a *location* parameter (the mean) and a *scale* parameter (characterizing the variability)—for example, the Normal density function and Binomial distribution. Parametric models have the advantage of simplicity (easy to estimate and interpret) but may have relatively high bias because real data may not obey the assumed functional form. The appendix contains a brief review of some of the more well-known parametric density and distribution models.
2. **Nonparametric Models:** where the distribution or density estimate is data-driven and relatively few assumptions are made a priori about the functional form. For example, we can use the kernel estimates introduced in chapter 3 and section 6.3.3: the local density at *x* is defined as a weighted average of points near to *x*.

Taking the above as the extremes, we can also define intermediate models that lie between these parametric and nonparametric extremes: *mixture models*. These are discussed below.

6.4.2 Mixtures of Parametric Models

A mixture density for **x** is defined as

(6.5) $$p(\mathbf{x}) = \sum_{k=1}^{K} p_k(\mathbf{x}|\theta_k)\pi_k .$$

This model decomposes the overall density (or distribution) for **x** into a weighted linear combination of *K component* or *class* densities (or distributions). Each of the component densities $p_k(\mathbf{x}|?_k)$ typically consists of a relatively simple parametric model (such as a Normal distribution) with parameters $?_k$. p_k represents the probability that a randomly chosen data point was generated by component *k*, $\sum_k \pi_k = 1$.

To illustrate, consider a single Normal distribution used as a model for a two-dimensional data set. This distribution can be thought of as a "symmetric bump function," whose location and shape we can to try to locate in the 2-space to model the density of the data as well as possible (see figure 6.6 for a simple example). An intuitive interpretation of the mixture model is that it allows us to place *k* of these bumps (or components) in the two-dimensional space to approximate the true density. The locations and shapes of the *k* bump functions can be fixed independently of each other. In addition, we are allowed to attach weights to the components. If the weights are positive and sum to 1 the overall function is still a probability density (see equation 6.5).

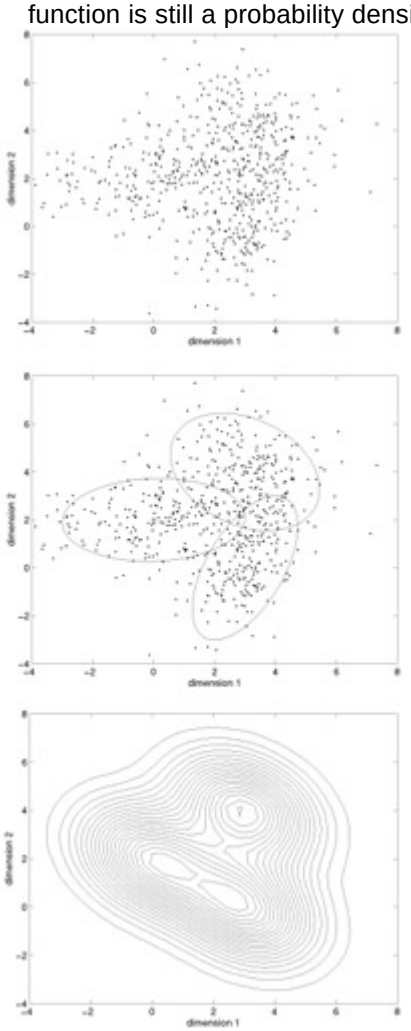

Figure 6.6: From the Top: (a) Data Points Generated From a Mixture of Three Bivariate Normal Distributions (Appendix 1) with Equal Weights, (b) the Underlying Component Densities Plotted as Contours that are Located 3s From the Means, and (c) the Resulting Contours of the Overall Mixture Density Function.

As *k* increases, the mixture model allows for quite flexible functional forms, as *local* bumps can be placed to capture local characteristics of the density (this is reminiscent of the local modeling ideas in regression). Clearly *k* plays the role of controlling complexity: for larger *k* we get a more flexible model but also one that it is more complicated to interpret and more difficult to fit. The usual bias-variance trade-offs again apply. Of course, we are not constrained to use only Normal components (although these tend to be quite popular in practice). Mixtures of exponentials and other densities could equally

well be used. The details of how the locations, shapes, and value of k are determined from the data are deferred until chapter 9. The important point here is that mixtures provide a natural generalization of the simple parametric density model (which is global) to a weighted sum of these models, allowing local adaptation to the density of the data in p-space.

The general principles underlying a mixture model are broadly applicable, and the general idea occurs in many guises in probabilistic model building. For example, the idea of hierarchical structure can be nicely captured using mixture models. In chapter 8 we will discuss the mechanics of how mixtures are fitted to data, and in chapter 9 we will see how they can be usefully employed for detecting clusters in data.

In terms of interpretability, either mixture models can be used simply as "black boxes" that provide a flexible model form, or the individual mixture components can be given an explicit interpretation. For example, components of a mixture model fitted to customer data could be interpreted as characterizing different types of customers. One interpretation of a mixture model (particularly in a clustering context) is that the components are generated by a *hidden variable* taking K values, and the location and shapes in p-space of the components are unknown to us a priori, but may be revealed by the data. Thus, mixture models share with projection pursuit and related methods the general idea of hypothesizing a relatively simple latent or hidden structure that may be generating the observed data. In chapter 8 and 10 we will discuss the use of the expectation-maximization (EM) algorithm for learning the parameters of mixture models from data.

6.4.3 Joint Distributions for Unordered Categorical Data

For categorical data we have a joint distribution function defined in the cross-product of all possible values of the p individual ariables. For example, if A is a variable taking values $\{a_1, a_2, a_3\}$ and B is a variable taking values $\{b_1, b_2\}$, then there are six possible values for the joint distribution of A and B. We will assume here (for simplicity) that the values are truly categorical and that there is (for example) no notion of scale or order. For small values of p, and for small numbers of variable values, it is convenient to display the values of the distribution in the form of a *contingency table* of cells, one cell per joint value, as shown in the example of table 6.1. This becomes impractical as the number of variables and values get beyond four or five. In addition, the contingency table does not really allow us to see any potential structure that might be in the data. For example, the data in table 6.1 have been constructed so that the variables are independent: however, this fact is not immediately apparent from looking at the table.

Table 6.1: A Simple Contingency Table for Two-Dimensional Categorical Data for a Hypothetical Data Set of Medical Patients Who Have been Diagnosed for Dementia.

		Dementia		
		None	Mild	Severe
Smoker	No	426	66	132
	Yes	284	44	88

In contrast to the case of quantitative variables, with categorical variables in which the categories are unordered there is no notion of a smooth probability function. Thus, if for example all variables each have m possible values, one would have to specify m^p - 1 independent probability values to specify the model fully (the -1 comes from the constraint that they sum to 1). Clearly this quickly becomes impractical as p and m increase. In the next section we look at systematic techniques for structuring both distribution and density functions to find parsimonious ways to describe high-dimensional data.

6.4.4 Factorization and Independence in High Dimensions

Dimensionality is a fundamental challenge in density and distribution estimation. As the dimensions of the **x**, space grow it rapidly becomes more difficult to construct fully specified model structures since model complexity tends to grow exponentially with dimension (the curse of dimensionality referred to earlier in this chapter).

Factorization of a density function into simpler component parts provides a general technique for constructing simple models for multivariate data. This is a simple yet powerful idea that recurs throughout multivariate modeling. For example, if we assume that the individual variables are *independent*, we can write the joint density function as

(6.6)
$$p(\mathbf{x}) = p(x_1, \ldots, x_p) = \prod_{k=1}^{p} p_k(x_k)$$

where $\mathbf{x} = (x_1, \ldots, x_p)$ and p_k is the one-dimensional density function for X_k. Typically it is much simpler to model the one-dimensional densities separately, than to model their joint density. Note that the independence model for log $p(\mathbf{x})$ has an *additive* form, reminiscent of the linear and additive model structures we discussed for regression. This factorization certainly simplifies things, but it has come at a modeling cost. The assumption that the variables are independent will not be even approximately true for many real problems. Thus, a full independence assumption is in essence one extreme end of a spectrum (the low-complexity end), a spectrum that extends to the fully specified joint density model at the other end (the high-complexity end). Of course, we do not have to choose models solely from the extremes of this complexity continuum, and can, instead, try to find something in between. The joint probability function $p(\mathbf{x})$ can be written in general as

(6.7)
$$p(\mathbf{x}) = p_1(x_1) \prod_{k=2}^{p} p(x_k | x_1, \ldots, x_{k-1})$$

The right-hand side factorizes the joint function into a sequence of conditional distributions. Now we can try to model each of those conditional distributions separately. Often considerable simplification results because each variable X_k is dependent on only a few of its predecessors. That is, in the conditional distribution for the kth variable, we can often ignore some of variables $X_1, \ldots, X_{k-1}$. Such factorizations permits a natural representation of the model as a directed graph, with the nodes corresponding to variables, and the edges showing dependencies between the variables. Thus the edges directed *into* the node for the kth variable will be coming from (a subset of) the variables $x_1, \ldots, x_{k-1}$. These variables are, naturally enough, called the *parents* of variable x_1. Sometimes we have to experiment by fitting different models to the data to seek such simplifying factorizations. In other cases such simplifications will be evident from the structure of the data—for example, if the variables represent the same property measured sequentially (for instance, at different times). In this case, a *Markov chain* model is often appropriate—in which all of the previous information relevant to the kth variable is contained in the immediately preceding variable (so that the terms in this factorization simplify to $p(x_k|x_1, \ldots, x_{k-1}) = p(x_k|x_{k-1})$). The model structure for a first-order Markov model is shown in figure 6.7.

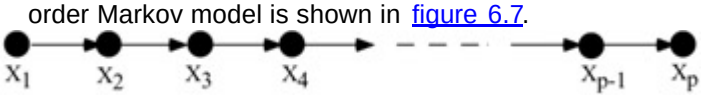

Figure 6.7: A Graphical Model Structure Corresponding to a First-Order Markov Assumption.

Graphs that are used represent probability models, such as that in figure 6.7 are often referred to as *graphical models*. In the discussion below we focus specifically on the widely-used subclass of acyclic directed graphs (also sometimes known in computer science as belief networks when used as probability models). Note that this graph representation emphasizes the independence structure of the model (e.g., see figure 6.7 again) and leaves the actual functional and numeric parametrization of parent-child relationships unspecified.

For another example of a graphical model, consider the variables age, education (level of education a person has) and baldness (whether a person is bald or not). Clearly age cannot depend on either of the other two variables. Conversely, both education and baldness are directly dependent on age. Furthermore, it is quite implausible that education and baldness are directly dependent on each other given age—that is, once we know the person's age, knowing whether or not they are bald tells us nothing about their education level (and vice versa). On the other hand, if we do not know a person's

age, then baldness may provide information about education (for example, a bald person is more likely to be older, and hence, in turn, more likely to have a university degree). Thus, a plausible graphical model is the one in figure 6.8.

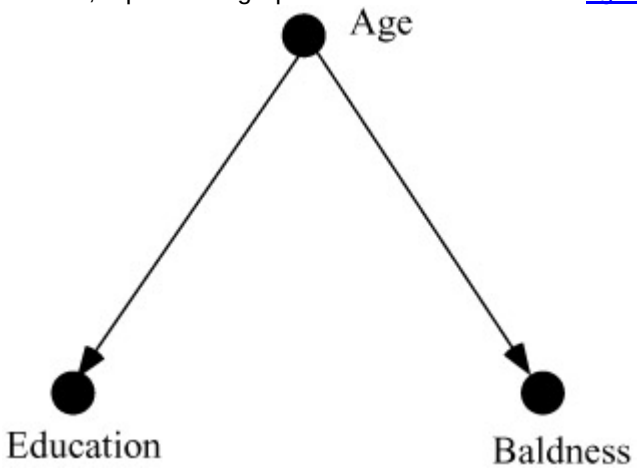

Figure 6.8: A Plausible Graphical Model Structure for Two Variables Education and Baldness that are Conditionally Independent Given Age.

These ideas can be taken further, by the postulation of the existence of unobserved hidden or *latent* variables, which explain many of the observed relationships in the data. Figure 6.9 provides such an example. In this model structure a single latent variable has been introduced as an intermediate variable that simplifies the relationship between the observed data (in this case, medical symptoms) and the underlying causal factors (here, two independent diseases). The introduction of hidden variables in a manner such as this can serve to simplify the relationships in a model structure; for example, given the values here of the intermediate variable, the symptoms become independent. However, we must exercise discretion in practice in terms of how many hidden variables we introduce into the model structure to avoid introducing spurious structure into the fitted model. In addition, as we will discuss in chapters 8 and 9, parameter estimation and model selection with hidden variables is quite nontrivial.

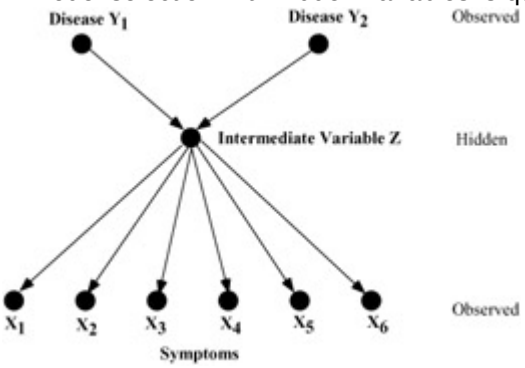

Figure 6.9: The Graphical Model Structure for a Problem with Two Diseases that are Marginally (Unconditionally) Independent, A Single Intermediate Variable *Z* that Directly Depends on Both Diseases, And Six Symptom Variables that are Conditionally Independent Given *Z*.

In the context of classification and clustering, it is often convenient to assume that the variables are *conditionally independent* of each other given the value of the class variable. That is,

(6.8) $$p(\mathbf{x}|y) = \prod_{j=1}^{p} p_j(x_j|y),$$

where *y* is a particular (categorical) class value. This is simply the conditional independence ("naive") Bayes model introduced in the context of classification modeling in section 6.3.5. The graphical representation for such a model is shown in figure 6.10.

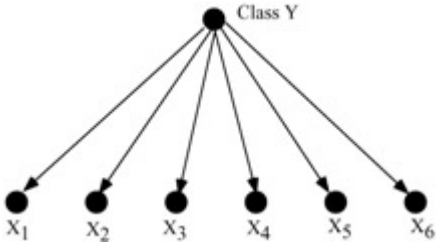

Figure 6.10: The First-Order Bayes Graphical Model Structure, With a Single Class *Y* and 6 Conditionally Independent Feature Variables $X_1, \ldots, X_6$.

Equation 6.8 can also be used in the case where *Y* is an unobserved (hidden, latent) variable that is introduced to simplify the modeling of $p(\mathbf{x})$, i.e., we have a finite mixture of the form

(6.9)
$$p(\mathbf{x}) = \sum_{k=1}^{K} \left(\prod_{j=1}^{p} p_j(x_j|y=k) \right) p(y=k),$$

where *Y* takes *K* values, and each component $p(\mathbf{x}|y=k)$ is modeled using the conditional independence assumption of equation 6.8. As an example, we might model the joint distribution of how customers purchase *p* products in this fashion, where (for example) if a customer belongs to a specific component *k* then the likelihood of purchasing certain subsets of products, i.e., $p_j(x_j|y=k)$, is increased for certain subsets of products x_j. Thus, although the products (the x_j) are modeled as being conditionally independent given $y = k$, the mixture model induces an unconditional (marginal) independence by virtue of the fact that certain products co-occur with higher probability in certain components *k*. In effect, the hidden *Y* variable acts to group the variables x_j together into equivalence classes, where within each equivalence class the variables are modeled as being conditionally independent. The use of hidden variables in this manner can be a powerful modeling technique, and it is one we return to in more detail in chapter 9.

6.5 The Curse of Dimensionality

We have noted in various places that what works well in a one-dimensional setting may not scale up very well to multiple dimensions. In particular, the amount of data we need often increases exponentially with dimensionality if we are to maintain a specific level of accuracy in our parameter or function estimates. This is sometimes referred to as the "curse of dimensionality." This can be important, since data miners are often interested in finding models and patterns in high-dimensional problems. Note that "high-dimensional" can be as few as $p = 10$ variables or as many as $p = 1000$ variables or beyond—it depends on the complexity of the models concerned and on the size of the available data.

Example 6.4

The following example is taken from Silverman (1986) and illustrates emphatically the difficulties of density estimation in high dimensions. Consider data that is simulated from a multivariate Normal density with unit covariance matrix and mean (0,0, . . . , 0) (see the appendix for a definition of the multivariate Normal density). Assume that the bandwidth *h*, in a kernel density estimate, is chosen such that it minimizes the mean square error at the mean. Silverman calculated the number of data points required to ensure that the relative mean square error at zero is less than 0.1—that is, that $E[(\hat{p}(\mathbf{x}) - p(\mathbf{x}))^2]/p(\mathbf{x})^2 < 0.1$—at $\mathbf{x} = \mathbf{0}$ and where $p(\mathbf{x})$ is the true Normal density, and $\hat{p}(\mathbf{x})$ is a kernel estimate using a Normal kernel with the optimal density, and bandwidth parameter. Thus, we are looking at the relatively "easy" problem of estimating (within 10% relative accuracy) a Normal density at the mode of this density (where the points will be most dense on average) by using a Normal kernel: what could be easier? Silverman showed (analytically) that the number of data points grows exponentially. In 1 dimension we need 4 points, in 2 we need 19, 3 we need 67, in 6 we need 2790, and by 10 dimensions we need about 842,000. This is an inordinate number of data points for such a simple problem! The lesson to be learned is that density estimation tasks (and indeed most other data mining tasks) rapidly become very difficult as dimensionality increases.

There are two basic (and fairly obvious) strategies for coping with high-dimensional problems. The first is simply to use a subset of relevant variables to construct the model. That is, to find a subset of p' variables where $p' << p$. The second is to transform the original p variables into a new set of p' variables, where again $p' << p$. Examples of this approach include of p principal component analysis, projection pursuit, and neural networks.

6.5.1 Variable Selection for High-Dimensional Data

Variable selection is a fairly general (and sensible) strategy when dealing with high-dimensional problems. Consider for example the problem of predicting Y using $X_1, \ldots, X_p$. It is often plausible that not all of the p variables are necessary for accurate prediction. Some X variables may be completely unrelated to the predictor variable Y (for example, the month of a person's birth is unlikely to be related to their creditworthiness). Others may be *redundant* in the sense that two or more X variables contain essentially the same predictive information. (For example, the variables income before tax and income after tax are likely to be highly correlated.)

We can use the notion of independence (introduced in chapter 3) to gauge *relevance* in a quantitative manner. For example, if $p(y|x_1) = p(y)$ for all values of y and x_1, then the target variable Y is independent of input variable X_1. If $p(y|x_1, x_2) = p(y|x_2)$, then Y is independent of X_1 if the value of X_2 is already known. In practice, of course, we are not necessarily able to identify from a finite sample which variables are independent and which are not; that is, we must estimate this effect. Furthermore, we are interested not only in strict independence or dependence, but also in the degree of dependence. Thus, we could (for example) rank individual X variables in terms of their estimated linear correlation coefficient with Y: that would tell us about estimated individual *linear* dependence. If Y is categorical (as in classification), we could measure the average mutual information between Y and X':

(6.10)
$$I(Y;X') = \sum_{i,j} p(y_i, x'_j) \log \frac{p(y_i, x'_j)}{p(y_i)p(x'_j)}$$

to provide an estimate of the dependence of X and Y, where X' here is a categorical variable (for example, a quantized version of a real-valued X). Other measures of the relationship between Y and the Xs can also be used.

However, the interaction of *individual* X variables with Y does not necessarily tell us anything about how *sets of variables* may interact with Y. The classic example, for Boolean variables, is the parity function, where Y is defined to be 1 if the sum of the (binary) values of the $X_1, \ldots, X_p$ variables in the set is an even integer, and Y is 0 otherwise. Y is independent of any individual X variable, yet is a deterministic function of the full set. While this is something of an extreme example, it nonetheless illustrates that such *non-linear non-additive* interactions can be masked if we only look at individual pair-wise interactions between Xs and Y. Thus, in the general case, the set of k *best individual* X variables (as ranked by correlation for example) is not the same as the *best set* of X variables of size k. Since one can have $2^p - 1$ different nonempty subsets of p variables, exhaustive search is not feasible except for very small p. Worse still, for many prediction problems, there is no optimal search algorithm (in the sense of being guaranteed to find the best set of variables) that has worst-case time complexity any better than $O(2^p)$.

This means that, in practice, subset selection methods tend to rely on heuristic search to find good model structures. Many algorithms are based on the simple heuristic of greedy selection, such as adding or deleting one variable at a time. We will return to this issue of search in chapter 8.

6.5.2 Transformations for High-Dimensional Data

The second general category of ideas is based on *transforming* the predictor variables. The intuitive idea here is to search for a set of p' variables (let us call them $Z_1, \ldots, Z_{p'}$),

where typically p' is much smaller than p, where the Z variables are defined as functions of the original X variables, and where the Zs are chosen in some sense to be the "best" set of p' variables for our task.

This general theme, of replacing the observed variables with a smaller set of variables that are somehow more fundamental to the task at hand, shows up repeatedly in different branches of data analysis. The Zs are variously referred to as *basis functions, factors, latent variables, principal components*, and so forth, depending on the specific goals and methods used to derive them. We will examine some of these models (and their associated fitting algorithms) in detail in later chapters, but for now we illustrate the general idea with just two specific examples:

- *Projection Pursuit Regression* uses a model structure of the form

(6.11) $$\hat{y} = \sum_{j=1}^{p'} w_j h_j(\alpha_j^T \mathbf{x})$$

- where $\alpha_j^T \mathbf{x}$ is the projection of the vector $\mathbf{x}$ onto the jth weight vector a_j (both vectors being p-dimensional, resulting in a scalar inner product), h_j is a nonlinear function of this scalar projection, and the w_j are scalar weights for the resulting nonlinear functions. The procedures for determining the w_j, the form of the h_j, and the "projection directions" a_j can be rather complex and algorithm-dependent, but the underlying idea is quite general.
- For example, this is essentially the form of the model structure that underlies neural networks (to be discussed later in chapter 11), where for such networks the functional forms of the h_j are usually chosen to be something like $h_j(t) = 1/(1 + e^{-t})$. One limitation of this class of models is the fact that they are quite difficult to interpret unless $p' = 1$. Another limitation is that the algorithms for estimating the parameters of these models can be computationally quite complex and may not be practical for very large data sets. We will return to this model family in chapter 11.
- *Principal Components Analysis*: We introduced principal components analysis (PCA) in chapter 3. This is a classic technique in which the original p predictor variables are replaced by another set of p variables ($Z_1, \ldots, Z_p$) that are formed from linear combinations of the original variables. The data vectors comprising the original data set map to new vectors in the $\mathbf{Z}$ space and, as explained in chapter 3, the sets of weights defining the Zs are chosen so as to maximize the variance of the original data set when expressed in terms of these new variables. Principal components analysis is thus a special case of projection pursuit, where the projection index in this case is the variance along the projected direction. Principal components has two merits as a data reduction technique. Firstly, it sequentially extracts most of the variance of the data in the X space, so we might hope that only the first few components (far fewer than the full number p of original X variables) contain most of the information in the data. Secondly, by virtue of the way in which the components are extracted (see chapter 3) they are orthogonal, so that interpretation is eased. However, one should be aware that the principal component vectors in the $\mathbf{X}$ space may not necessarily be the ideal projection directions for optimizing *predictive* performance on a different variable Y (for example). For example, when we try to model differences among groups (or classes) in the data (for classification and clustering), the principal component projections need not emphasize group differences and indeed can even hide them. (Similar remarks can be made about more general projection pursuit methods.) Nonetheless, PCA is widely used in data analysis and can be a very useful dimension-reduction tool. There are a wide number of other techniques (each with different properties) available for dimension reduction, including factor analysis (chapter 4), projection pursuit (chapter 11, and above), independent component analysis, and so forth.

6.6 Models for Structured Data

In many situations either the individuals, the variables, or both, possess some well-defined relationships that are known a priori. Examples include linear chains or sequences (where the measurements are ordered—for example, protein sequences), time series (where the measurements are ordered in time, perhaps on a uniform time scale), and spatial or image data (where the measurements are defined on a spatial grid). Even more complex structure is possible. For example, in medicine one can have imaging data of the brain measured on a three-dimensional grid, with repeated measurements over time.
Such structured data is inherently different from the types of measurements we have discussed in most places in this chapter. Up to this point we have implicitly assumed that the n individual objects (the patients, the customers) in our data set are a random sample from an underlying population. Specifically, we have assumed that the measurement vectors $\mathbf{x}(i)$, $1 = i = n$, are conditionally independent of each other given a particular fitted model (that is, that the likelihood of the data can be expressed as the product of individual $p(\mathbf{x}(i))$. For example, if we have a Normal density model for the variable weight, then we are assuming that knowing the weight of one person tells us nothing about the weight of any other person in the data set. (We are, of course, here ignoring subtle dependencies that may exist such as having members of the same family appear sequentially in our data set, where such family members might be predisposed to having similar overweight or underweight tendencies.) Thus, although it may be an approximation, we have been working with this assumption on the basis that it is a useful assumption for many practical situations.

However, there are problems for which the dependence is explicit and needs to be modeled. For example, if we take measurements of a person's blood pressure every five minutes over a 24-hour period, then clearly there is very likely to be some significant dependence between the successive values. How should we model such dependence? One approach is to reduce the multiple observations on each object to one or a few variables (that is, a fixed multivariate description $\mathbf{x}$), using ideas about the expected relationships between them (we referred to this possibility above). This is sometimes called the *feature extraction* approach. For example, we might expect blood pressure to decrease over the 24-hour period as a medication begins to take effect, so we might replace the 5 times 12 times 24 observations for each person by just two numbers showing a starting value and the decreasing slope of a linear trend. Or we might use the same principle and fit a curve in which the rate of decrease reduces over time. The numbers describing the curves for each subject (which are often called *derived variables*) can then be analyzed in the standard way.

Note that this general approach (of converting sequential measurements into a non-sequential vector representation) may be sufficient for a given data mining task, but in general there is a loss of information in this process, in that we lose the timing and order information present in the original measurements. For certain applications this sequential information may be critical. As an example, we may have a population of Web users, among whom are a group who navigate from Web page A, to page B, to page C, repeatedly in that order, in a cyclic fashion. If we were to reduce this information to a histogram of which pages were visited (yielding a histogram with three roughly equal bins), we would lose the ability to discover the dynamic cyclic pattern underlying the data.
Let us consider an example of a sequential data model, namely a first-order Markov model for T data points observed sequentially, $y_1, \ldots, y_T$. Note that for even moderately large values of T, a full joint density for $p(y_1, y_2, \ldots, y_T)$ will be a very complex object (for example, if Y takes m discrete values, it will require the specification of $O(m^T)$ numbers). Thus, in modeling data with structure, we can take direct advantage of the ideas presented in the last section on factorization; that is, the structure of the data will suggest a natural structuring for any models we will build. Thus, we return to our first-order Markov model, again defined as:

(6.12) $$p(y_1, \ldots, y_T) = p_1(y_1) \prod_{t=2}^{T} p_t(y_t | y_{t-1})$$

We can simplify this model considerably if we make the assumption of *stationarity*, namely that the probability functions in the model do not depend on the specific time *t*, that is, $p_t(y_t|y_{t-1}) = p(y_t|y_{t-1})$. Thus, the *same* conditional probability function is used in different parts of the sequence. This drastically cuts down on the number of parameters we need for the model. For example, if *Y* is *m*-ary, the nonstationary model would require $O(m^2T)$ parameters (a matrix of $m \times m$ conditional probabilities for each time point in the sequence), while the stationary model only requires $O(m^2)$ probabilities (one matrix of $m \times m$ conditional probabilities that is used throughout the sequence). The notion of stationarity can be applied to much more general Markov models than the first-order model above, and indeed extends naturally to spatial data models as well (for which we would assume stationarity in space, rather than in time). If we assume stationarity, then we cannot account for *changes* in the statistical model as a function of time or space. However, stationarity is advantageous from a parametrization standpoint, making it a very useful and practical assumption in model building—we will assume it throughout our discussion unless specifically stated otherwise.

The Markov model in equation 6.12 has a simple generative interpretation (see figure 6.7, with *y*s instead of *x*s). The first value in the sequence y_1 is chosen by drawing a y_1 value randomly according to some initial distribution $p(y_1)$. The value at time $t = 2$ is randomly chosen according to the conditional density function $p(y_2|y_1)$, where the value y_1 is known and fixed. Once y_2 has been chosen in this manner, y_3 is now generated according to $p(y_3|y_2)$ where the value y_2 is now fixed, and so on until time *T*.

However, the Markov model assumption is rather strong (as we discussed in section 6.4.4). In words, it says that the influence of the past is completely summarized by the value of *Y* at time *t*-1. Specifically, Y_t does not have any "long-range" dependencies other than its immediate dependence on Y_{t-1}. Clearly there are many situations in which this model may not be accurate. For example, consider modeling the grammatical structure of English text, where *Y* takes values such as `verb, adjective, noun`, and so on. The first-order Markov assumption is inadequate here since (for example) deciding whether a verb is singular or plural will depend on the subject of the verb, that in turn may be much further back in the sentence than just one word back.

For real-valued *Y* s, the Markov model is often specified as a conditional Normal distribution:

(6.13) $$p(y_t|y_{t-1}) = \frac{1}{\sqrt{2\pi}\sigma} exp - \frac{1}{2}\left(\frac{y_t - g(y_{t-1})}{\sigma}\right)^2$$

where $g(y_{t-1})$ plays the role of the mean of the Normal (it is a deterministic function linking the past y_{t-1} to the present y_t) and s is the noise in the model (assumed stationary here). A common choice for the function *g* is to make it a linear function of y_{t-1}, $g(y_{t-1}) = a_0 + a_1 y_{t-1}$, leading to the well-known *first-order autoregressive model*,

(6.14) $$y_t = \alpha_0 + \alpha_1 y_{t-1} + e$$

where *e* is zero-mean Gaussian noise with standard deviation s and the as are the parameters of the model. Note that equation 6.14 can be expressed in the form of equation 6.13 under these assumptions.

The model in equation 6.14 has a simple interpretation from a generative viewpoint; the value y_t at time *t* in the sequence is generated by taking the previous value y_{t-1}, multiplying it by a constant a_1, adding an offset a_0, and adding some amount of random noise *e*. For *y* to remain stable (bounded as *t* ? 8) it is necessary that $-1 < a_1 < 1$. Values of $|a_1|$ closer to 1 imply stronger dependence among successive *y* values; values of $|a_1|$ closer to 0 imply weaker dependence. This model structure is clearly closely related to the standard regression model structures of section 6.3. Instead of regressing on independent *X* values, here *Y* is regressed on "lagged" values of itself. Thus, from our knowledge of regression model structures, we can immediately think of a multitude of generalizations of the simple first-order model above. For example, y_t can depend on earlier lags in the sequence; that is, we can replace the mean at time *t* by $g(y_{t-1})$ in equation 6.13 with $g(y_{t-1}, y_{t-2}, \ldots, y_{t-k})$, known as a *kth order Markov model*. Again, a common choice for $g(y_{t-1}, y_{t-2}, \ldots, y_{t-k})$ is a simple linear model of the form $a_0 + S\, a_i y_i$. In principle, however, rather than just linear regression, we could use any of the general

functional forms discussed in section 6.3, such as additive models, polynomial models, local linear models, data-driven local models, and so forth.

A further important generalization of the Markov model structures we have discussed so far is to explicitly model the notion of a hidden *state variable*. The general notion of hidden state for sequential and spatial models is prevalent in engineering and the sciences and recurs in a variety of functional model forms. Specific examples of such structures include hidden Markov models (HMMs) and Kalman filters. The HMM structure is easily explained by looking at its corresponding graphical model structure, shown in figure 6.11. From a generative viewpoint a first-order HMM operates as follows (picture the observations being generated by moving from left to right along the chain). The hidden state variable X is categorical (corresponding to m discrete states) and is first-order Markov. Thus, x_t is generated by sampling a value from the conditional distribution function $p(x_t|x_{t-1})$ in the usual Markov chain fashion, where $p(x_t|x_{t-1})$ is an $m \times m$ matrix of conditional probabilities. Once the state at time t is generated (with value x_t), an observation y_t is now generated with probability $p(y_t|x_t)$. Here y_t could be univariate or multivariate, or real-valued or categorical, or a combination of both. Thus, in a HMM, the observations y_t only depend on the state at time t, and the state sequence is a first-order Markov chain. The state sequence is unobserved or hidden, and the ys are directly observed: thus, there is uncertainty (given a model structure and a set of observed y's) about which particular state sequence generated the data.

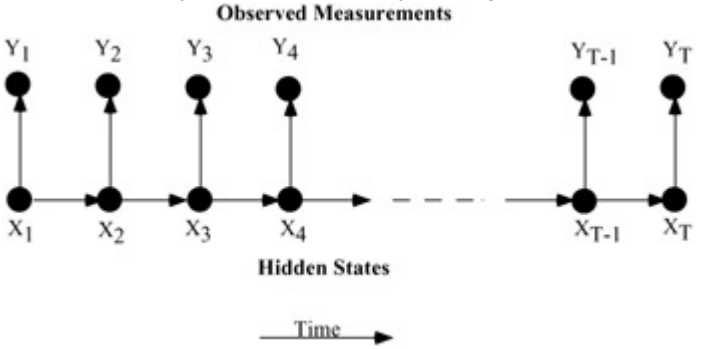

Figure 6.11: A Graphical Model Structure Corresponding to a First-Order Hidden Markov Assumption.

We can think of the HMM structure as a form of mixture model (m different density functions for the Y variable), where we have now added Markov dependence between "adjacent" mixture components x_t and x_{t+1}. For the record, the joint probability of an observed sequence and any particular hidden state sequence for a first-order HMM can be written as:

(6.15)
$$p(y_1, \ldots, y_T, x_1, \ldots, x_T) = p(x_1)p(y_1|x_1)\prod_{t=2}^{T} p(y_t|x_t)p(x_t|x_{t-1}).$$

The factorization on the right-hand side is apparent from the graphical model structure in figure 6.11. When regarded as a function of the parameters of the distributions, this is the likelihood of the variables $(Y_1, \ldots, Y_T, X_1, \ldots, X_T)$. The likelihood of the observed ys is useful for fitting such model structures to data (that is, learning the parameters of $p(y_t|x_t)$ and $p(x_t|x_{t-1})$). To calculate $p(y_1, \ldots, y_T)$ (the likelihood of the *observed* data) one has to sum the left-hand side terms over the m^T possible state sequences, that appears at first glance to involve a sum over an exponential number of terms. Fortunately there is a convenient recursive way to perform this calculation in time proportional to $O(m^2T)$.

Again, it is clear that we can generalize the first-order HMM structure in different directions. A kth order Markov model corresponds to having x_t depend on the previous k states. The dependence of the ys can also be generalized, allowing for example y_t to have a linear dependence on the k previous ys (as in an autoregressive model) as well as direct dependence on x_t. This yields a natural generalization of the usual autoregressive model structure to a *mixture of autoregressive models*, which we can think of generatively as switching (in Markov fashion) among m different autoregressive models. *Kalman filters* are a closely related cousin of the HMM, where now the hiddenstates are *real-valued* (such as the unknown velocity or momentum of a vehicle, for example), but the independence structure of the model is essentially the same as we have described it for an HMM.

Computer scientists will recognize in our generative description of a hidden Markov model that it is quite reminiscent of a finite state machine (FSM). In fact, as we have described it here, a first-order HMM is directly equivalent to a stochastic FSM with m states; that is, the choice of the next state is governed by $p(x_{t+1}|x_t)$. This naturally suggests a generalization of model structures in terms of different *grammars*. Finite-state machines are simple forms of grammar known as *regular grammars*. The next level up (in the socalled Chomsky hierarchy of grammars) is the *context-free grammar*, which can be thought of as augmenting the finite-state machine with a *stack*, permitting the model structure to "remember" long-range dependencies such as closing parentheses at the ends of clauses, and so forth. As we ascend the grammar hierarchy, our model structures become more expressive, but also become much more difficult to fit to data. Thus, despite the fact that regular grammars (or HMMs) are relatively simple in structure, this form of model structure has dominated the application of Markov models to sequential data (over other more complex grammar structures), due to the difficulties of fitting such complex structures to real data.

Finally, although we have only described simple data structures where the Y s exist in an ordered sequence, it is clear that for more general data dependencies (such as data on a two-dimensional grid) we can think of equivalent generalizations of the Markov model structures to model such dependence. For example, *Markov random fields* are essentially the multidimensional analogs of Markov chains (for example, in two dimensions we would have a grid structure rather than a chain for our graphical model). It turns out that such models are much more difficult to analyze and work with than chain models. For example, problems such as summing out the hidden variables in the likelihood (as we mentioned for equation 6.15) do not typically admit tractable solutions and must be approximated. Thus, spatial data can be more difficult to work with than sequential data, although conceptually the ideas of stationarity, Markovianity, linear models, and so forth, can all still be applied. One common approach with gridded data, which may or may not make sense depending on the application, is to "shape" the two-dimensional grid data (say $n \times n$ grid points) into a single vector of length n^2, perform PCA on these vectors, project each set of grid measurements onto a small set of PCA vectors, and model the data using standard multivariate models in this reduced dimensional space. This approach ignores much of the inherent spatial information in the original grid, but nonetheless can be quite practical in many situations. Similarly, for *multivariate time series or sequences*, where we have p different time series or sequences measured over the same time frame (corresponding for example to different biomedical monitors on the same patient), we can use PCA to reduce the p original time series to a much smaller number of "component" series for further analysis.

6.7 Pattern Structures

Throughout this book, we have characterized a model as describing the whole (or a large part of the) data set, and a pattern as characterizing some local aspect of the data. A pattern can be considered to be a predicate that returns `true` for those objects or parts of objects in the data for which the pattern occurs, and `false` otherwise. To define a class of patterns we need to specify two things: the syntax of the patterns (the language specifying how they are defined) and their semantics (our interpretation of what they tell us about data to which they are applied). In this section we consider patterns for two different types of discrete-valued data: data in standard matrix form and data described as strings.

6.7.1 Patterns in Data Matrices

A generic approach for building patterns is to start from primitive patterns and combine them using logical connectives. (An alternative is to build a special class of patterns for a particular application.) Returning again to our data matrix notation, assume we have p variables $X_1, \ldots, X_p$. Let $\mathbf{x} = (x_1, \ldots, x_p)$ be a p-dimensional vector of measurements of these variables. We denote the ith individual in the data set as $\mathbf{x}(i)$, where $1 = i = n$. The entire data set $D = \{\mathbf{x}(1), \ldots, \mathbf{x}(n)$. In turn, $x_k(i)$ is the value of the kth measurement on the ith individual.

In general, a *pattern* for the variables $X_1, \ldots, X_p$ identifies a subset of all possible observations over these variables. A general language for expressing patterns can be built by starting from *primitive patterns*. These are simply conditions on the values of the variables. For example, if c is a possible value of X_k, then $X_k = c$ is a primitive pattern. If the values of X_k are ordered (for example, numbers on the real line), we can also include inequalities such as $X_k = c$ as primitive conditions. If needed, the primitive patterns could also include multivariate conditions such as $X_k X_j > 2$ for numeric data or $X_k = X_j$ for discrete data.

Given a set of primitive patterns, we can form more complex patterns by using logical connectives such as *AND* (?) and *OR* (V). For example, we can form a pattern

(age = 40) ? (income = 10)

that describes a certain subset of the input records in a payroll database. Note, for example, that each branch of a classification tree (as described in chapter 5) forms a conjunctive pattern of this form. Another example is the pattern

(chips = 1) ? (beer = 1 V soft drink = 1)

describing a subset of rows in a market basket database.

A *pattern class* is a set of legal patterns. A pattern class C is defined by specifying the collection of primitive patterns and the legal ways of combining primitive patterns. For example, if the variables $X_1, \ldots, X_p$ all range over {0,1}, we can define a class of patterns C consisting of all possible conjunctions of the form

$$(X_{j_1} = 1) \wedge (X_{j_2} = 1) \wedge \cdots \wedge (X_{j_k} = 1).$$

Patterns in this class that occur frequently in a data set D are called *frequent sets* (of variables), since each such pattern is uniquely determined by a sub-set of the variables: this pattern could be written just as $X_{i_1}, X_{i_2}, \ldots, X_{i_k}$. Conjunctive patterns such as frequent sets are relatively easy to discover from data, and we consider them in detail in chapter 13.

Given a pattern class and a dataset D, one of the important properties of a pattern is its frequency in the data set. The frequency $fr(?)$ of a pattern ? can be defined as the relative number of observations in the dataset about which ? is true. In some cases, only patterns that occur reasonably often are of interest in data mining. However, having a frequency of a pattern close to 0 can also be quite informative in its own right. (Indeed, sometimes it is the rare but unusual pattern that is of particular interest.) Of course, the frequency of a pattern is not the only important property of the pattern. Properties such as semantic simplicity, understandability, and the novelty or surprise of the pattern are obviously also of interest. As an example, for any particular observation $(x_1, \ldots, x_p)$ in the data set we can write a conjunctive pattern $(X_1 = x_1) ? \ldots ? (X_p = x_p)$ that matches exactly that observation. The disjunction of all of such conjunctive patterns forms a pattern that has frequency 1 for the data set. However, the pattern would be just a bloated way of writing out the entire data set and would be quite uninteresting.

Given a class of patterns, *a pattern discovery task* is to find all patterns from that class that satisfy certain conditions with respect to the data sets. For example, we might be interested in finding all the frequent set patterns whose frequency is at least 0.1 and where the variable X_7 occurs in the pattern. More generally, the definition of the pattern discovery task might include also conditions on the informativeness, novelty, and understandability of the pattern. In defining the pattern class and the pattern discovery task the challenge is to find the right balance between expressivity of the patterns, their comprehensibility, and the computational complexity of solving the discovery task.

Given a class of patterns C, we can easily define rules. A *rule* is simply an expression ? ? ? , where ? and ? are patterns from a pattern class C. The semantics of a logical rule are that if the expression ? is true for an object, then ? is also true. We can relax this definition to allow for uncertainty in the mapping from ? to ? , where ? is true with some probability if ? is true. The *accuracy* of such a rule is defined as $p(? | ?)$, the conditional probability that ? is true for an object, given that ? is true. As is described in chapter 4, we can easily estimate such probabilities from a data set using appropriate frequency counts; that is

$$p(\varphi|\rho) = \frac{fr(\rho \wedge \varphi)}{fr(\rho)}.$$

The *support* $fr(? \; ? \; ? \;)$ of the rule $? \; ? \; ? \;$ can be defined either as $fr(?)$ (the fraction of objects to which the rule applies) or $fr(?? \; ?)$ (the fraction of objects for which both the left and right-hand sides of the rule are true).

For example, if our patterns are frequent sets, then a rule would have the form

$$\{A_1, \ldots, A_k\} \; ? \; \{B_1, \ldots, B_h\}.$$

where each of the A_ks and B_js are binary variables. Written out in full, the rule would be

$$(A_1 = 1) \; ? \; \ldots \; ? \; (A_k = 1) \; ? \; (B_1 = 1) \; ? \; \ldots \; ? \; (B_h = 1).$$

Such rules are called *association rules*, a widely used pattern structure in data mining (we will discuss in detail the algorithmic principles behind finding such rules in chapter 13).

Here we have described patterns that define subsets of the original data set. That is, each pattern was defined by a formula that referred only to the variables of a single observation. In certain cases, however, we need to use patterns defined by referring to several observations. For example, we might wish to identify all those points in a geographical database that form the vertices of an equilateral triangle. As a more formal example, consider a data set with discrete variables $A_1, \ldots, A_p$. A *functional dependency* is an expression of the form

$$A_{i_1} A_{i_2} \cdots A_{i_k} \rightarrow A_{i_{k+1}},$$

where $1 = i_j = p$ for $i = 1, \ldots, k+ 1$. Note the syntactic similarity to the definition of association rules. However, the *functional dependency* defined by this expression is true in a data set if, for all pairs of observations $\mathbf{x} = (a_1, \ldots, a_p)$ and $\mathbf{y} = (b_1, \ldots, b_p)$ in the data set, we have that if $\mathbf{x}$ and $\mathbf{y}$ agree on all the variables A_{i_j} for $j = 1, \ldots, k$, then $\mathbf{x}$ and $\mathbf{y}$ agree also on $A_{i_{k+1}}$. That is, if $a_{i_j} = b_{i_j}$ for all $i = 1, \ldots, k$, then also $a_{i_{k+1}} = b_{i_{k+1}}$. Functional dependencies have their roots in database design, and they are also of interest in query optimization. Knowing the functional dependencies that hold in a data set may be important for understanding the structure of the data.

The patterns or conditions written in these refer only to the values occurring in a single record in the database. Sometimes we are also interested in describing patterns that refer to other observations, such as those that arise in "the employees whose income is the smallest in their department." Such conditions can also be described using logical formalisms. For example,

$$\{\mathbf{x}_k \mid \text{age} = 40 \; ? \; \text{income} = 10\}.$$

6.7.2 Patterns for Strings

In the last section we discussed examples of patterns for data in the traditional matrix form. Other types of data require other types of patterns. To illustrate, we consider patterns for strings. Formally, a *string* over alphabet S is a sequence $a_1 \ldots a_n$ of elements (also called *letters*) of S. The alphabet S can be the binary alphabet {0,1}, the set of all ASCII codes, the DNA alphabet {A,C,G,T}, or the set of all words consisting of ASCII characters. The set of all strings built from letters from S is denoted by S^*.

Note how string data differs from data in standard matrix form: for a string, there is no fixed set of variables. If and when we want to use the notions of probability to describe string data, we typically consider each of the letters of the string to be a random variable. The data can be one or several strings, and in most cases we are interested in finding out how many times a certain pattern occurs in the strings. (For example, we might want to compute the number of exact occurrences of a certain DNA sequence in a large collection of sequences.) The simplest string pattern is a substring: the pattern $b_1 \ldots b_k$ occurs in the string $a_1 \ldots a_n$ at position i, if $a_{i+j-1} = b_j$ for all $j = 1, \ldots, k$. For example, for DNA sequences we might be interested in finding occurrences of the substring pattern ATTATTAA, and for strings over the ASCII alphabet we might be interested in whether or not the pattern `data mining` occurs in a given string.

For strings we might, however, be interested in a larger class of patterns. A *regular expression E* is an expression that defines a set $L(E)$ of strings. The expression E is one of

1. a string s; then $L(s) = \{s\}$
2. a concatenation E_1E_2; in this case the set $L(E_1E_2)$ consists of all strings that are a concatenation of a string in $L(E_1)$ and a string in $L(E_2)$
3. a choice $E_1 \mid E_2$; then $L(E_1 \mid E_2) = L(E_1) \cup L(E_2)$
4. an iteration E^*; then $L(E^*)$ consists of all strings that can be written as a concatenation of 0 or more strings from $L(E)$

Thus, 10(00|11)*01 is a regular expression that describes all strings that start with 10 and end with 01 and in between contain a sequence of pairs 00 and 11.

Regular expressions are a form of patterns that are quite well suited to describing interesting classes of strings. While there are simple classes of strings that cannot be described by regular expressions (such as the set of strings consisting of all balanced sequences of parentheses), many quite complicated phenomena of strings can still be captured by using them.
While regular expressions are fine for defining patterns over strings, they are not sufficiently expressive for expressing variations in the occurrence times of events. A simple class of patterns that can take the occurrence times into account is the *episode*. At a high level, an episode is a partially ordered collection of events occurring together. The events may be of different types, and may refer to different variables. For example, in biostatistical data an event might be a headache followed by a sense of disorientation occurring within a given time period. It is also useful for them to be insensitive to intervening events—as with, for example, alarms in a telecommunications network, logs of user interface actions, and so on. Episodes can also be incorporated into the type of rules discussed earlier.

6.8 Further Reading

There are many books on regression modeling. Draper and Smith (1981) and Cook and Weisberg (1999) each provide excellent overviews. McCullagh and Nelder (1989) is the definitive text on generalized linear models and Hastie and Tibshirani (1990) is equally definitive on generalized additive models. Fan and Gijbels (1996) provide a very extensive discussion of local polynomial methods and Wand and Jones (1995) contains a more theoretically oriented treatment of kernel estimation methods (both for regression and density estimation). Hand (1982) contains a detailed description of kernel methods in supervised classification problems.
Fairly recent treatments of advances in classification modeling are provided by McLachlan (1992), Ripley (1996), Bishop (1996), Mitchell (1996), Hand (1997), and Cherkassky and Muller (1998). The McLachlan text and the Ripley text are aimed primarily at a statistical audience. A notable feature of Ripley's text is the illustration of basic concepts using a variety of different data sets. The Bishop text and the Cherkassky and Muller texts are more focused on neural networks and related developments, and each contains many ideas that have yet to make their way into the mainstream statistical literature. Duda and Hart (1973) remains a classic in the classification literature with a very clear and comprehensive treatment of essential ideas involved in building classification models. Reviews of Bishop (1996), Ripley (1996), Looney (1997), Nakhaeizadeh and Taylor (1997), and Mitchell (1997) are provided in *Statistics and Computing*, volume 8, number 1.
The most comprehensive texts on mixture models are those of Titterington, Makov, and Smith (1985) and McLachlan and Basford (1988), and McLachlan and Peel (2000). Other general discussions of the area include Redner and Walker (1984) and Everitt and Hand (1981). Silverman's 1992 text on density estimation contains a wealth of insight, while Scott's (1992) text on the same topic is notable for its discussion of "average-shifted histogram" models that share some of the properties of both histograms and kernel estimators, and that might be of interest for models based on "binning" of massive data sets.