the resurgence of interest during the 1980s, after development of the extended methods we discuss in later chapters. Much more recently, Mooney, Shavlik, Towell, and Gove (1989) have shown that even simple methods like PCP perform quite well in many domains, although researchers now typically focus on more sophisticated techniques.

Breiman, Friedman, Olshen, and Stone (1984) first described the IWP algorithm for determining weights on threshold units, which they used as a subroutine in constructing multivariate decision trees (see Chapter 7), and Murthy, Kasif, Salzberg, and Beigel (1993) have explored related methods in a similar context. We did not discuss linear discriminant analysis (Fisher, 1936), another class of methods for finding linear splits, that is widely used in statistics. Work on spherical units, such as that by Moody and Darken (1991) and Kruschke (1992), has typically been done within the framework of multilayer inference networks (see Chapter 6).

Criteria tables have their origin in medicine, where they are commonly used to diagnose diseases. Spackman (1988) first introduced them to the machine learning literature, but they have received little attention there except for work by Murphy and Pazzani (1991), Baffles and Mooney (1993), and a few others. We introduced them here not because they are historically important but because they provide a useful transition between logical and threshold approaches to induction.

CHAPTER 4

The Induction of Competitive Concepts

In the first chapter we argued that competitive concepts differ from both the logical and threshold schemes in relying on a best-match interpreter. Here we consider some instantiations of this approach, which originated within the field of pattern recognition and has only recently been adopted by the machine learning community. The representational power of competitive concepts is similar to that of threshold concepts, and one can even adapt threshold formalisms to operate in a best-match fashion, but we will not consider them here.

Instead, we will focus on two broad approaches to the induction of competitive concepts. The first involves the storage of specific instances or prototypes in memory, whereas the second forms abstract descriptions cast as probabilistic summaries. In each case, we initially consider generic issues in the representation and use of such concepts, then introduce some induction methods that operate on these representations.

The task confronting competitive induction methods is much the same as that we have considered in earlier chapters. Given a set of preclassified training instances, they must find some intensional description – in this case a set of prototypes or probabilistic summaries – that correctly classifies novel test instances. Note the use of the plural here. Unlike the logical and threshold frameworks, competitive schemes cannot represent concepts in isolation; the extension of a concept depends not only on its own description but on that of others as well. For this reason, competitive learning methods always acquire multiple concepts rather than individual ones.

The typical learning operator for competitive methods involves the computation of an average over the training instances. The resulting

description specifies the central tendency of the instances that produced it. Typically, averaging does not lead to strictly more general or more specific concepts; its effect is more akin to the parameter setting used with threshold concepts. However, even the latter can be usefully viewed in terms of search through a space of concept descriptions.

In contrast, most mechanisms used for the induction of competitive concepts are so simple that the search metaphor hardly seems appropriate. Some variations process one instance at a time, and these constitute examples of incremental hill-climbing methods in that they retain a single summary description in memory and carry out no explicit backtracking. However, they are not driven by errors; the actions taken by the learner are independent of whether the performance element makes the correct classification. For this reason, incremental versions typically produce the same results as nonincremental ones; there are no effects of training order.

4.1 Instance-based learning

As we mentioned above, one approach to inducing competitive concepts diverges from the methods we examined in previous chapters by storing specific instances rather than abstract descriptions. There exist a variety of learning algorithms that follow this basic approach. When applied to classification tasks, as we examine in this chapter, they are usually called *instance-based* or *nearest neighbor* methods. Here we will examine only the simplest form of these methods, but we will encounter more sophisticated versions in Chapter 5. When used for problem solving and related tasks, they are more commonly referred to as *case-based* or *analogical* methods, which we discuss in Chapters 9 and 10.

4.1.1 The representation and use of prototypes

As usual, we will first address issues of representation and performance before turning to the learning methods themselves. Instance-based methods represent each concept description in terms of *prototypes*, which are simply conjunctions of attribute-value pairs. In Chapter 5, we will see versions of this scheme that represent each class with multiple prototypes, but here we will restrict our attention to representations that allow only one prototype per class.

The instance-based approach is most easily visualized for domains that involve numeric attributes, so we will use one of these for our examples. Figure 4-1 (a) depicts prototypes for two classes in the numeric domain from previous chapters, which involves the `height` and `girth` of a person. Class `A` has an average `height` of 5 feet and an average `girth` of 6 feet. In contrast, the `height` of class `B` is 8 feet and its `girth` is 3 feet. The prototypes include no information about the distribution of these dimensions; they store only specific values.

Because they use the same representational language as instances, it is easy to confuse such prototypes with extensional definitions. However, they are just as much intensional definitions as the logical and threshold concepts in previous chapters, and one must combine them with a well-specified interpreter to determine their extension and decision boundaries.

Using prototypes for classification is a three-step process. Given a new instance I, one first computes its distance from each prototype. For numeric domains, this is often the Euclidean distance in the space of instances defined by the domain's attributes. Thus, the distance between an instance x and prototype p is

$$\sqrt{\sum_{i=1}^{a}(p_i - x_i)^2} \quad ,$$

where a is the number of attributes. For nominal domains, one would typically use a variant on the Hamming distance, in which the distance varies inversely with the number of features that agree in two instances. In some cases, researchers instead refer to the *similarity* between two instances, which is usually just the inverse of their distance. Researchers have explored a variety of metrics for use in calculating distance.

After computing the distance between the instance and each description, one uses the result to classify the instance. The simplest scheme simply predicts the class associated with the nearest prototype. For example, suppose we encounter a person we would like to assign to class `A` or `B` in Figure 4-1 (a), and whose `height` is 6 feet and `girth` is 2 feet. This description's Euclidean distance to the `A` prototype is 2.24, whereas the distance to the `B` summary is 4.12. Thus, one would assign the person to the closer `A` class.

The figure shows the decision boundary formed by the two prototypes. All instances closer to the `A` prototype fall into its decision region,

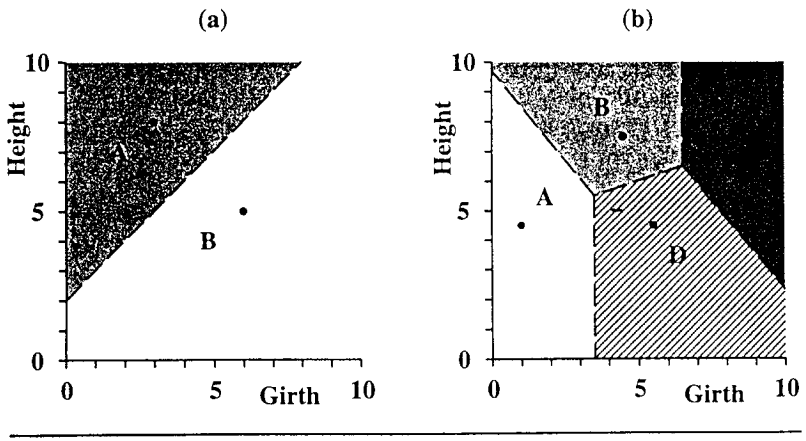

Figure 4-1. Prototypes (black points) and decision boundaries (dashed lines) for (a) a two-class domain and (b) a four-class domain.

whereas all instances closer to B (including our example) contribute to its decision region. Figure 4-1 (b) presents a more general situation that involves four classes. In general, there will exist one such decision boundary for each pair of classes C_1 and C_2, with each point on the hyperplane being equidistant from the prototypes for C_1 and C_2. Thus, k classes lead to $k(k-1)/2$ hyperplanes, although some of these will have no effect on classification. In our four-class example, only five of the six potential boundaries appear, since the boundary between A and C is redundant.

Clearly, classification to the nearest prototype gives an effect very similar to that of the linear threshold units we considered in the previous chapter. In fact, there exists a straightforward mapping from a prototype representation to a linear threshold unit. For two classes a and b, the decision boundary between these classes can be described by the linear threshold unit

$$(p_{a,1} - p_{b,1})x_1 + \ldots + (p_{a,i} - p_{b,i})x_i > \frac{1}{2}\sum_j (p_{b,j}^2 - p_{a,j}^2) \quad ,$$

where $p_{k,j}$ is the value of the prototype for class k on attribute x_j and the term on the right side of the inequality is the threshold term. For

example, the prototypes in Figure 4-1 (a) give us

$$(6-3)\ \texttt{girth} + (5-8)\ \texttt{height} > \frac{1}{2}\cdot[(36+25)-(9+64)] \quad ,$$

which we can rewrite as the inequality

$$3\cdot\texttt{girth} - 3\cdot\texttt{height} > -6 \quad .$$

Instances meeting this constraint[1] are assigned to class B, with others going to A. In the multi-class case, one must check all pairs of inequalities that separate the classes, then assign the instance to the one for which it satisfies all threshold expressions.

The above mapping holds for numeric domains, but it does not tell the full story. In Boolean and attribute-value domains, instance-based methods typically use non-Euclidean measures of distance, which makes the relation to linear threshold units less clear. But in all cases, the combination of prototypes with a competitive matching scheme produces decision boundaries that divide the instance space into convex regions, as seen in our examples.

4.1.2 An instance-averaging method

Perhaps the simplest algorithm for learning competitive concept descriptions relies on a process that averages the training instances to find the prototype for each class. Thus, to determine the value p_i for a prototype along numeric attribute i, one computes

$$p_i = \frac{1}{n}\sum_{j=1}^{n} x_{ij} \quad ,$$

where x_{ij} is the value on attribute i of the jth of n instances of a given class. For reasons we will discover, this technique is uncommon, but it provides a good introduction to methods for the induction of competitive concepts.

An incremental variant of this algorithm begins with no prototypes in memory. Upon encountering an instance for a novel class, it simply stores that instance in memory along with the associated class. When it observes a new instance in the same class, the method averages the

1. Dividing this expression by 3 gives a line with slope 1 and intercept of 2, which accurately describes the decision boundary in the figure.

description for the instance into the prototype. In particular, for each numeric attribute i, the change in the prototype's value is

$$\Delta p_i = \frac{x_i - p_i}{n+1} \quad ,$$

where p_i is the stored value, x_i is the observed value, and n is the number of cases encountered for the class before the new instance. Thus, the incremental-averaging algorithm must store the instance count in addition to the average, even though the performance component uses only the latter. If some instances omit the values of certain attributes, then it must store different counts for each.

Figure 4-2 (a) shows the prototypes and decision boundaries for two classes, A and B, along with a training instance for the former. If we assume that the A prototype is based on two instances, then Figure 4-2 (b) depicts the revised prototype that results from averaging the new case into the description for the A class. The computation is straightforward: the new average for **height** is $8.0 + \frac{9.5-8.0}{3} = 8.5$, whereas the new average for **girth** is $3.5 + \frac{6.5-3.5}{3} = 4.5$. Furthermore, the count for this class would be incremented to 3. The extension of the B class has changed even though its intensional description has remained unaltered; as we noted earlier, the extension of a competitive concept C is a function not only of C's description, but also of its competitors'.

For nominal attributes, the prototypical value is simply the modal value, i.e., the one that occurs most frequently for a given class. Thus, the incremental version of instance averaging must retain even more statistics for nominal features, keeping counts for each possible value. This lets it replace the current modal value with another if the latter's count comes to exceed that for the former. For example, given the training instances **dark thin**, **dark thick**, and **light thick**, the modal values would be **dark** and **thick**. However, this could change if later instances contain the values **light** or **thin**.

In terms of efficiency and elegance, it is difficult to imagine an induction algorithm that is superior to the instance-averaging method. However, this comes at a severe cost in representational and learning power. As we have seen, the decision boundaries that the method implicitly constructs are equivalent to those formed by linear threshold units, and our discussion in Chapter 3 noted the representational limits of single hyperplanes. Briefly, they can handle concepts that are linearly separable and they can approximate concepts that cover contiguous regions of

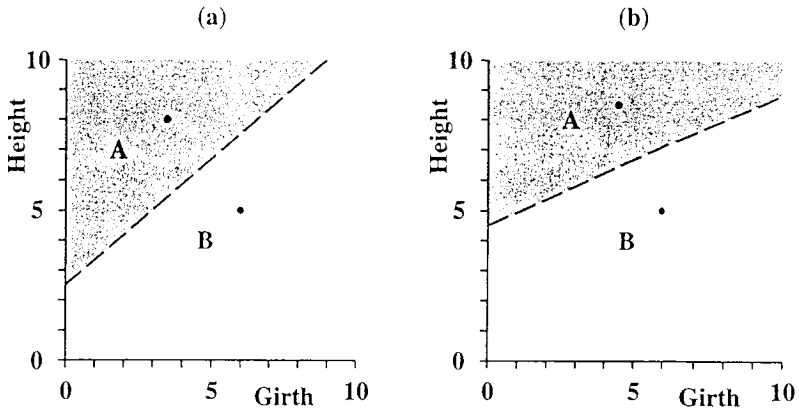

Figure 4-2. Prototypes (black points) and decision boundary (dashed line) for two classes, A and B, (a) before averaging a training instance for class A (white point), and (b) the modified prototype and decision boundaries after averaging. This example assumes that the A prototype in (a) is based on two instances.

the instance space, but they cannot produce the decision regions needed for concepts that are not singly connected. Nevertheless, hyperplanes give surprisingly accurate results on many classification tasks, despite their simplicity.

Yet the ability to *represent* arbitrary hyperplanes does not translate into an ability to *learn* such decision boundaries. The hyperplanes constructed by the instance-averaging method are completely determined by the average values observed for each class. Unlike the perceptron learning algorithm, it cannot alter weights and thresholds in response to errors; one can only hope that the 'weights' produced by the averaging process will form a useful decision boundary between the classes. Although the literature reports no experimental results on this method, it seems unlikely that its behavior would compare favorably to even such simple methods as the PCP or LMS algorithms.

4.1.3 An attribute-weighting method

One drawback of the instance-averaging method is its treatment of all attributes as equivalent. This can cause problems if different attributes involve different measurement scales (e.g., feet and miles) or even if they

only have different ranges of values. One can easily sidestep these issues by noting the minimum and maximum values for each attribute, then normalizing the values to fall between 0 and 1. Incremental versions of this technique have some sensitivity to training order, but the basic idea seems sound.

However, the underlying problem with simple instance averaging runs deeper. In many domains, only some attributes are relevant to the target concept, but even with normalization, each attribute has the same importance in the distance computation used in the classification decision. Thus, a test instance might match the prototype for the correct class well on the relevant attributes but be assigned to the prototype for another class because it happens to match the latter well on irrelevant attributes. Learning can eventually correct such problems, but only after the method has observed many additional training instances.

Figure 4-3 (a) illustrates this problem in a variant of the height/girth domain in which only `girth` is relevant. The figure shows three training instances for each of two classes, the prototype that they produce for each class, and the decision boundary that results, given a Euclidean distance metric. Because the prototype for class `A` happens to be greater on the `height` dimension than the prototype for class `B`, the learned boundary diverges considerably from the target boundary. Moreover, the prototypes actually misclassify some of the training instances. Such problems tend to disappear with larger training sets, since they are more likely to contain representative samples, but it gets worse with increasing numbers of irrelevant attributes.

One response to this problem is to modify the distance metric used during classification so that it gives different weights w_i to each attribute A_i, producing the revised distance metric

$$\sqrt{\sum_i w_i (p_i - x_i)^2} \quad ,$$

where x_i is the value of the test instance on attribute i and p_i is a prototype's value on the same attribute. This metric gives the effect of shrinking the instance space more along some dimensions than others. For example, assigning `height` a weight very close to 0 would cause the classification process to effectively ignore this attribute, giving a decision boundary that is nearly parallel to the irrelevant axis.

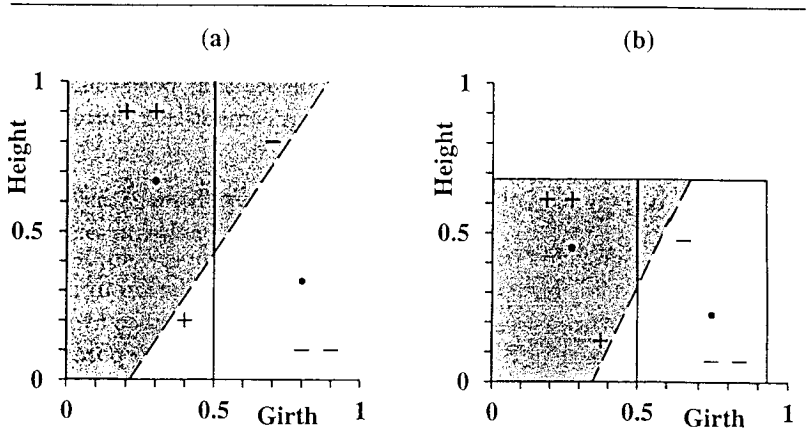

Figure 4-3. Training instances (+ and –), prototypes (black points), and decision boundaries (dashed line) for two classes, (a) using equal weights on the `height` and `girth` attributes and (b) giving the weight 0.68 to `height` and 0.93 to `girth`. The effect of weighting reduces the range of values differently for each attribute, thus altering the slope of the decision boundary. The target boundary (solid line) is parallel to the axis of the irrelevant attribute `height`.

Naturally, one would like to learn an appropriate set of weights from the training data. A simple scheme of this sort uses the equation

$$w_i = 1 - \frac{1}{n} \sum_{k=1}^{c} \sum_{j=1}^{n_k} |p_{ki} - x_{ji}|$$

to determine the weight w_i, where n is the total number of training instances, n_k is the number of training instances for class k, p_{ki} is the value for the prototype for class k on attribute i, and x_{ji} is the normalized value of the jth instance on that attribute. Given the normalized training data in Figure 4-3 (a), this expression produces the modified weights 0.68 for `height` and 0.93 for `girth`. Figure 4-3 (b) shows the resulting decision boundary, which is closer to the target boundary and which correctly classifies all of the training instances.

Many alternative methods exist for modifying the weights used during instance-based classification. For example, some incremental algorithms modify weights only on the basis of misclassified training instances.

Also, one can easily adapt the scheme described above to determine separate weights for each class, and one can also take into account the relative frequency of each class. We will soon see that both ideas play a central role in another approach to inducing competitive concepts.

4.2 Learning probabilistic concept descriptions

Probabilistic approaches to induction are somewhat more complex than instance-based ones, but they also have a firmer foundation in mathematics. The basic framework is similar in spirit to the instance-averaging method we discussed in Section 4.1.2, except that it incorporates information about the *distribution* of classes and their attribute values, and the induction process estimates this distributional information from training cases. Below we examine the representation, use, and acquisition of such probabilistic summaries.

4.2.1 Probabilistic representation and classification

Like the methods we considered in the previous section, the basic probabilistic scheme associates a single description with each class, but it stores probabilistic information about the class rather than a prototype. In particular, each description has an associated class probability or base rate, $p(C_k)$, which specifies the prior probability that one will observe a member of class C_k. Each description also has an associated set of conditional probabilities, specifying a probability distribution for each attribute.

In nominal and Boolean domains, one typically stores a discrete probability distribution for each attribute in a description. Each $p(v_j|C_k)$ term specifies the probability of value v_j, given an instance of concept C_k. In numeric domains, one must represent a *continuous* probability distribution for each attribute. This requires that one assume some general form or model, typically the normal distribution. Conveniently, a given normal curve can be represented entirely in terms of its mean μ and its variance σ^2. Moreover, the probability for any given numeric value v can be easily determined from

$$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-(v-\mu)^2/2\sigma^2} \quad ,$$

which is the probability density function for the normal distribution.

To classify a new instance I, one can apply Bayes' theorem to determine the probability of each class C_i given the instance,

$$p(C_i|I) = \frac{p(C_i)p(I|C_i)}{p(I)} \quad .$$

However, since I is simply a conjunction of j values, we can expand this to the expression

$$p(C_i|\bigwedge v_j) = \frac{p(C_i)p(\bigwedge v_j|C_i)}{\sum_k p(\bigwedge v_j|C_k)p(C_k)} \quad ,$$

where the denominator sums over all classes and where $p(C_i|\bigwedge v_j)$ is the probability that the instance I is a member of class C_i given value v_j. After calculating this probability for each description, one assigns the instance to the class with the highest probability. A performance system that uses this approach is generally known as a *Bayesian classifier*.

However, in order to make the above expression operational, one must still specify how to compute the term $p(\bigwedge v_j|C_k)$. Most versions of Bayesian classifiers assume independence of attributes, which lets one calculate $p(C_i|I)$ using the equality

$$p(\bigwedge v_j|C_k) = \prod_j p(v_j|C_k) \quad ,$$

where the values $p(v_j|C_k)$ represent the conditional probabilities stored with each class. This approach is sometimes called a *simple* or *naive* Bayesian classifier. Although it seems unlikely that the independence assumption is perfectly satisfied in natural domains, the extent to which a Bayesian classifier's behavior depends on this assumption remains an open issue.

Consider the probabilistic summaries from the three-attribute cell domain depicted in Figure 4-4 (a). The `healthy` and `virulent` classes have equal base rates of $\frac{1}{2}$, but their attribute distributions are quite different. The conditional probability of observing `one nucleus` in a `healthy` cell is $\frac{2}{3}$, whereas the conditional probability for `two nuclei` is only $\frac{1}{3}$. The odds are also 2 to 1 that a `healthy` cell will have `one tail` rather than `two tails`, and that it will have `light color` rather than `dark color`. The probability distributions for the `virulent` class are exactly inverted, with `two nuclei` expected $\frac{2}{3}$ of the time, `one nucleus` in only $\frac{1}{3}$ of the cases, and so forth.

(a)

P(HEALTHY) = 1/2		P(V|HE)
NUCLEI	ONE	2/3
	TWO	1/3
TAILS	ONE	2/3
	TWO	1/3
COLOR	LIGHT	2/3
	DARK	1/3

P(VIRULENT) = 1/2		P(V|VIR)
NUCLEI	ONE	1/3
	TWO	2/3
TAILS	ONE	1/3
	TWO	2/3
COLOR	LIGHT	1/3
	DARK	2/3

(b)

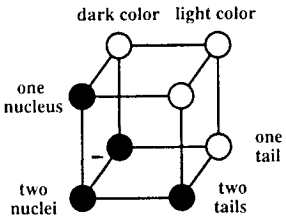

Figure 4-4. (a) Probabilistic summaries for the classes `healthy` and `virulent` from the three-attribute cell domain. (b) The decision regions produced by these summaries in combination with a simple Bayesian classifier that assumes attributes are independent, with white circles indicating `healthy` cells and black ones `virulent` cells.

Now consider the calculations involved in classifying a new cell that has `one tail`, `one nucleus`, and `light color`. If the Bayesian classifier makes the independence assumption, then for each class it finds the conditional probabilities for each observed value, and multiplies them by each other and by the base rate. For the `healthy` class, this gives $\frac{2}{3} \times \frac{2}{3} \times \frac{2}{3} \times \frac{1}{2} = \frac{4}{27}$; for the `virulent` class, it instead gives $\frac{1}{3} \times \frac{1}{3} \times \frac{1}{3} \times \frac{1}{2} = \frac{1}{54}$. The sum of these two products is $\frac{1}{6}$, and dividing each by this amount gives a probability of $\frac{8}{9}$ that the cell is `healthy` and only $\frac{1}{9}$ that it is `virulent`. The Bayesian classifier selects the more likely of these two as its prediction, which makes sense, since the instance perfectly matches the modal values for the `healthy` class.

Figure 4-4 (b) shows the decision regions produced by a Bayesian classifier for these two probabilistic summaries, which looks suspiciously like those we saw in Chapter 3. In fact, for domains that involve Boolean features, there exists a direct mapping from the Bayesian classifier representation to an equivalent linear threshold unit. In particular, the weight w_i for a given attribute i is computed as

$$w_i = \log \left[\frac{p(x_i|C)(1 - p(x_i|\bar{C}))}{p(x_i|\bar{C})(1 - p(x_i|C))} \right] ,$$

where $p(x_i|C)$ is the probability of feature x_i given the class C and $p(x_i|\bar{C})$ is the probability of that feature given the other class, and the threshold w_0 for the linear unit becomes

$$w_0 = \log \left[\frac{p(C)}{1 - p(C)} \right] + \sum_{j=1}^{i} \log \left[\frac{1 - p(x_i|C)}{1 - p(x_i|\bar{C})} \right] .$$

Note that, in this framework, the base rate comes into play only in the threshold term, which makes intuitive sense. A generalized version of this mapping holds for multi-class induction tasks, which Bayesian classifiers can also handle.

However, it would be wrong to state that the probabilistic scheme has exactly the same representational power as linear threshold units, for they can produce quite different effects in numeric domains. Figure 4-5 illustrates the variety of decision regions that can emerge with Bayesian classifiers when one assumes that, in addition to being independent, the attributes are normally distributed. For example, Figure 4-5 (a) shows a situation in which the variances for one class are smaller than those for another on both attributes. Bayes' rule dictates that an instance I near the mean of the tighter class is more likely to come from this class, but for more distant instances, the probability of the other class is higher. The result is an elliptical decision region for the tighter class.

Another effect occurs when the classes have different variances on one attribute but the same on another, as in Figure 4-5 (b). Here the outcome is a parabolic decision region for the tighter class, since instances to the right of the two means are always more likely to come from that class. In some cases, a Bayesian classifier can even produce decision regions that are not singly connected, as in Figures 4-5 (c) and (d). Here one class has less variance on one attribute, whereas the other class is tighter on the second. The result here depends on the placement of the means, giving a hyperbolic region in one case and an hourglass-shaped region in the other.

4.2.2 Induction of naive Bayesian classifiers

Learning in Bayesian classifiers is a simple matter, very similar to the instance-averaging process we described earlier. The basic process can operate either incrementally or nonincrementally, but since the order of training instances has no effect on learning, the results are identical. The simplest implementation increments a count each time it encounters a new instance, along with a separate count for a class each time it is given an instance of that class. Together, these counts let the classifier estimate $p(C_k)$ for each class C_k.

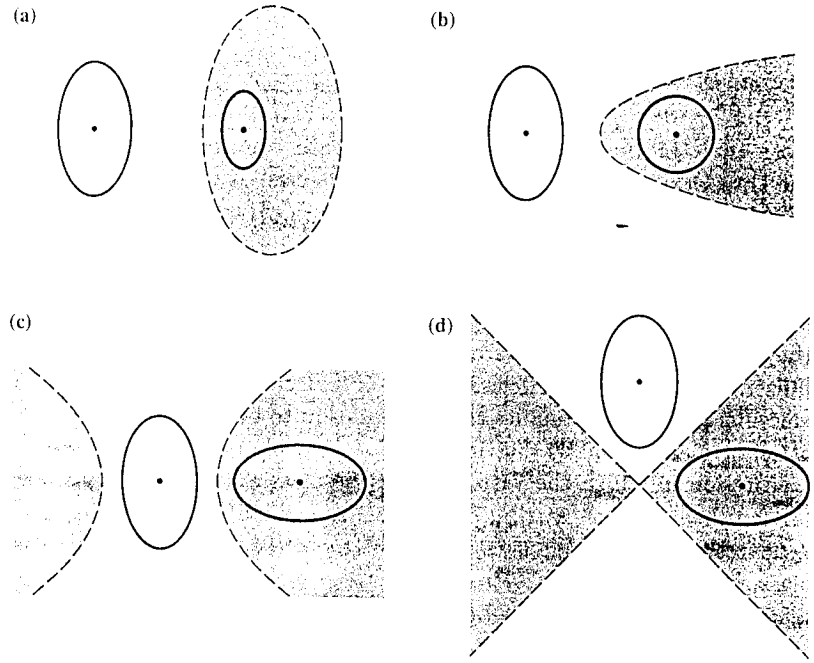

Figure 4-5. Forms of decision regions that a Bayesian classifier can produce for the two-class case in a two-attribute numeric domain, if one assumes that the attributes are independent and normally distributed. Different means (shown by dots) and variances (shown by ellipses that indicate constant probability density) can produce: (a) an elliptical region; (b) a parabolic region; (c) hyperbolic regions; and (d) hourglass regions. After Duda and Hart (1973).

In addition, for each instance of a class that has a given nominal value, the algorithm updates a count for that class-value pair. Together with the second count, this lets the classifier estimate $p(v_j|C_k)$. For each numeric attribute, the method retains and revises two quantities, the sum and the sum of squares, which let it compute the mean and variance for a normal curve that it uses to find $p(v_j|C_k)$. In domains that can have missing attributes, one must include a fourth count for each class-attribute pair.

For example, suppose the probabilistic summaries in Figure 4-4 (a) are based on three training instances for each class. Given a new `healthy` cell that has `one tail`, `one nucleus`, and `light color`, the Bayesian classifier would increment its overall count, its class count, and its count

for each observed value. The resulting description for `healthy`, expressed in ratios, would include a base rate of $\frac{4}{7}$ and conditional probabilities of $\frac{3}{4}$ for `one tail`, `one nucleus`, and `light color`. The probabilities for alternative values would decrease to $\frac{1}{4}$, just as the base rate for `virulent` would lower to $\frac{3}{7}$. Taken together, these changes can affect the decision boundary and thus the classes to which cases are assigned.

The effects of learning are more complex in numeric domains. For example, observing enough instances near the mean of the rightmost class in Figure 4-5 (a) can reduce the variance enough to transform the decision region from an ellipse to the parabola in (b). Similarly, seeing enough instances that diverge from this mean along the horizontal attribute can transform (a) into the hyperbolic decision region in (c), whereas enough observations of the leftmost class above its mean can lead regions like those in (a) to evolve into ones like those in (d). Such phase changes can greatly affect the accuracy of the induced Bayesian classifier.

In some cases, the training data may produce a 0 for a base rate or a conditional probability. Since the classification decision involves multiplication, this overwhelms the effects of other factors. One way to avoid this problem is to replace 0 entries with $\frac{1}{n}$, where n is the number of training examples. When the Bayesian classifier has observed few instances, this gives a conditional probability that is relatively high, but if the class or value still does not appear after many training cases, the $\frac{1}{n}$ factor causes the estimate to approach 0.

Another response to this issue, more in accordance with Bayesian principles, is to initialize the class descriptions based on 'prior' probabilities. If the user has expectations about the base rates, conditional probabilities, or means and variances, he can set these at the outset. Lacking such knowledge, one can use 'uninformed priors' to initialize the probabilistic summaries. For example, given three classes, it seems reasonable, lacking other information, that they will each occur $\frac{1}{3}$ of the time; similarly, given a Boolean feature, one might assume $\frac{1}{2}$ as its conditional probability.

The use of priors forces one to decide how much influence to give them relative to the observations. A widespread solution assigns a constant α_j to each value v_j of a nominal attribute, then uses the expression

$$P(v_j|C_k) = \frac{c_j + a_j}{n_k + \sum_i \alpha_i} \quad ,$$

where c_j is the number of instances of class C_k in which value v_j has appeared, n_k is the number in which the attribute has occurred, and the sum occurs over values of the attribute. A common ploy involves setting all α_j to 1, which gives the prior probabilities the same influence as a single training instance. Analogous techniques exist for numeric attributes.

4.2.3 Comments on Bayesian classifiers

We have seen that the simple Bayesian classifier relies on two important assumptions. First, it posits that one can characterize each class with a single probabilistic description. In Boolean domains, this is equivalent to assuming linear separability, although the story is more complex for nominal and numeric attributes. Nevertheless, like perceptrons and simple instance-averaging methods, Bayesian classifiers are typically limited to learning classes that occupy contiguous regions of the instance space. In the next chapter, we will consider methods that move beyond this limitation.

Second, the simple Bayesian classifier also assumes independence of the attributes, which can cause problems on data like those in Figure 4-6 from the four-attribute cell domain. In this case, two attributes (`color` and `thickness`) are perfectly correlated and thus redundant; this is one extreme form of attribute dependence. Given only the features `nuclei`, `tails`, and `color`, a simple Bayesian classifier can induce a set of descriptions for the two classes that successfully label the training set; these descriptions predict `virulent` if any two of three attribute values are present. However, if the instances are equally likely, the algorithm cannot manage this feat when all four attributes are included. Effectively, the redundancy of `color` and `thickness` gives these features more influence than they deserve, causing the learned descriptions to classify the second training case as `virulent` rather than `healthy`, and no amount of training can overcome this error.

One alternative to the independence assumption stores the conditional probabilities for combinations of attribute-value pairs. In principle, given a attributes, one could estimate all a-way combinations of values, thus ensuring independence of 'features'. However, this is equivalent to storing the conditional probability for *every possible instance*. Even independent of its storage requirements, such a scheme (sometimes called an *optimal* Bayesian classifier) is impractical because it requires implausible numbers of observations to estimate parameters. In Chapter 6, we consider a more selective approach to using higher-order features in probabilistic induction.

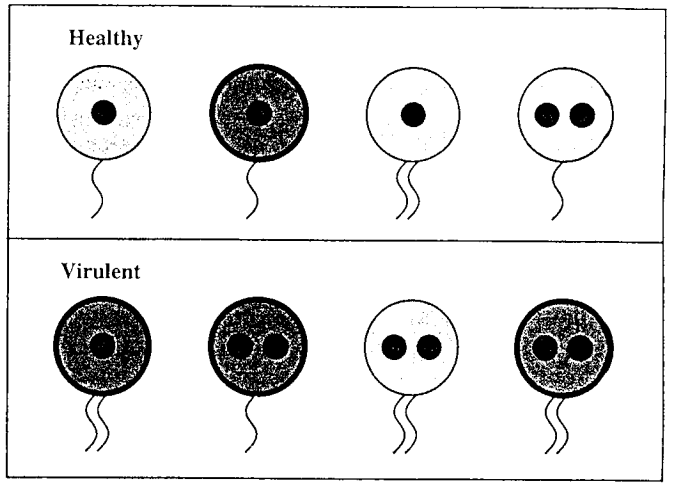

Figure 4-6. Training cases involving four attributes, two of which are redundant. These instances cannot be fully discriminated by a simple Bayesian classifier, which labels the second `healthy` cell as `virulent` after learning.

The two strong assumptions that underlie the simple Bayesian classifier have given it a poor reputation, but the implications of these assumptions for behavior on real-world domains remain an open question. The literature contains results for Bayesian classifiers on natural domains, where one would expect neither assumption to hold, that are comparable to those obtained for the more sophisticated methods we consider in later chapters. Moreover, the probabilistic representation and classification procedure of Bayesian classifiers should make them inherently robust in noisy domains, and also should give them an ability to discriminate relevant attributes from irrelevant ones. We hope that future research will better determine how this algorithm fares under different conditions.

4.3 Summary of the chapter

In this chapter we explored methods for inducing two generic classes of competitive concepts. These approaches share a number of basic ideas: maintaining a single description for each class, using a best-match interpreter to classify instances, and employing some form of averaging during the induction process. The resulting decision boundaries are very similar to those formed with linear threshold units, and in many cases there exists a simple mapping from one representation to the other.

The first competitive approach – instance-based learning – represents each class as a prototype or central tendency, and assigns an instance to the class having the prototype that is nearest according to some distance measure. Such methods typically induce the prototype for a class through simple averaging of the descriptions for training cases of that class. However, this approach does not scale well to domains that involve many irrelevant attributes, and in response we considered an alternative scheme that learns weights on attributes for use in the distance metric.

The second approach we examined – the Bayesian classifier – represents each class in terms of probability distributions, including information about the base rates and conditional probabilities. The performance component uses Bayes' rule to compute the probability of each class given an instance description, typically assuming that attributes are independent, and assigns the instance to the most likely class. The learning algorithm simply updates the distributional information by incrementing counts for classes and for attribute values observed in the training instances. Thus, Bayesian classifiers also learn weights on attributes, although these have clear probabilistic semantics.

Both instance-based and probabilistic algorithms can operate either incrementally or nonincrementally, the former constituting examples of incremental hill-climbing methods. Although simple in nature, these techniques produce respectable results on many domains, and they provide another basis for more sophisticated algorithms that can use them as subroutines, as we will see in the next chapter.

Exercises

1. Use the instance-averaging method (Section 4.1.2) to compute the prototypes for the training data in Figure 3-4 from Chapter 3. Draw the positions of these prototypes and the decision boundaries that they produce. Also show the decision boundaries that would occur if each of the four instances were prototypes for a separate class.
2. Transform the first pair of prototype descriptions from Exercise 1 into a single linear threshold unit, as described in Section 4.1.1.
3. Use the attribute-weighting method from Section 4.1.3 to alter the distance metric for the data in Exercise 1. State the weight found for each attribute and compute the distance of both prototypes from a test case with 3.0 `girth` and 5.0 `height`. Draw the warped instance space, as in Figure 4-3 (b), and the revised decision boundary.
4. Show the probabilistic summaries induced by a naive Bayesian classifier from the eight instances in Figure 3-1 (a) from Chapter 3. Show the instances that the resulting summaries classify correctly and incorrectly. Also show the probability calculations involved in classifying the instance `two tails`, `one nucleus`, and `thick wall`. Repeat this process for the instances in Figure 3-1 (c).
5. Treat the attributes in Exercise 4 as Boolean features, and transform each set of probabilistic summaries generated there into an equivalent linear threshold unit, as described in Section 4.2.1.
6. Compute the means and variances produced by a naive Bayesian classifier on the numeric training data in Figure 3-4. Draw the means and the approximate decision boundary that results. Explain why this boundary differs from those found in the first two exercises.

Historical and bibliographical remarks

The induction of competitive concepts has long been studied in the fields of pattern recognition and statistics, and general treatments can be found in Nilsson (1965) and in Duda and Hart (1973). They have also played a role in cognitive psychology, where they have served as models of human concept learning. Reviews of these exemplar (instance-based) and independent cue (naive Bayesian) models appear in Smith and Medin (1981) and in Fisher and Langley (1990).