# CHAPTER 8

# INSTANCE-BASED LEARNING

In contrast to learning methods that construct a general, explicit description of the target function when training examples are provided, instance-based learning methods simply store the training examples. Generalizing beyond these examples is postponed until a new instance must be classified. Each time a new query instance is encountered, its relationship to the previously stored examples is examined in order to assign a target function value for the new instance. Instance-based learning includes nearest neighbor and locally weighted regression methods that assume instances can be represented as points in a Euclidean space. It also includes case-based reasoning methods that use more complex, symbolic representations for instances. Instance-based methods are sometimes referred to as "lazy" learning methods because they delay processing until a new instance must be classified. A key advantage of this kind of delayed, or lazy, learning is that instead of estimating the target function once for the entire instance space, these methods can estimate it locally and differently for each new instance to be classified.

## 8.1 INTRODUCTION

Instance-based learning methods such as nearest neighbor and locally weighted regression are conceptually straightforward approaches to approximating real-valued or discrete-valued target functions. Learning in these algorithms consists of simply storing the presented training data. When a new query instance is encountered, a set of similar related instances is retrieved from memory and used to classify the

new query instance. One key difference between these approaches and the methods discussed in other chapters is that instance-based approaches can construct a different approximation to the target function for each distinct query instance that must be classified. In fact, many techniques construct only a local approximation to the target function that applies in the neighborhood of the new query instance, and never construct an approximation designed to perform well over the entire instance space. This has significant advantages when the target function is very complex, but can still be described by a collection of less complex local approximations.

Instance-based methods can also use more complex, symbolic representations for instances. In case-based learning, instances are represented in this fashion and the process for identifying "neighboring" instances is elaborated accordingly. Case-based reasoning has been applied to tasks such as storing and reusing past experience at a help desk, reasoning about legal cases by referring to previous cases, and solving complex scheduling problems by reusing relevant portions of previously solved problems.

One disadvantage of instance-based approaches is that the cost of classifying new instances can be high. This is due to the fact that nearly all computation takes place at classification time rather than when the training examples are first encountered. Therefore, techniques for efficiently indexing training examples are a significant practical issue in reducing the computation required at query time. A second disadvantage to many instance-based approaches, especially nearest-neighbor approaches, is that they typically consider *all* attributes of the instances when attempting to retrieve similar training examples from memory. If the target concept depends on only a few of the many available attributes, then the instances that are truly most "similar" may well be a large distance apart.

In the next section we introduce the *k*-NEAREST NEIGHBOR learning algorithm, including several variants of this widely-used approach. The subsequent section discusses locally weighted regression, a learning method that constructs local approximations to the target function and that can be viewed as a generalization of *k*-NEAREST NEIGHBOR algorithms. We then describe radial basis function networks, which provide an interesting bridge between instance-based and neural network learning algorithms. The next section discusses case-based reasoning, an instance-based approach that employs symbolic representations and knowledge-based inference. This section includes an example application of case-based reasoning to a problem in engineering design. Finally, we discuss the fundamental differences in capabilities that distinguish lazy learning methods discussed in this chapter from eager learning methods discussed in the other chapters of this book.

## 8.2 *k*-NEAREST NEIGHBOR LEARNING

The most basic instance-based method is the *k*-NEAREST NEIGHBOR algorithm. This algorithm assumes all instances correspond to points in the $n$-dimensional space $\Re^n$. The nearest neighbors of an instance are defined in terms of the standard

Euclidean distance. More precisely, let an arbitrary instance $x$ be described by the feature vector

$$\langle a_1(x), a_2(x), \ldots a_n(x) \rangle$$

where $a_r(x)$ denotes the value of the $r$th attribute of instance $x$. Then the distance between two instances $x_i$ and $x_j$ is defined to be $d(x_i, x_j)$, where

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

In nearest-neighbor learning the target function may be either discrete-valued or real-valued. Let us first consider learning discrete-valued target functions of the form $f : \Re^n \rightarrow V$, where $V$ is the finite set $\{v_1, \ldots v_s\}$. The $k$-NEAREST NEIGHBOR algorithm for approximating a discrete-valued target function is given in Table 8.1. As shown there, the value $\hat{f}(x_q)$ returned by this algorithm as its estimate of $f(x_q)$ is just the most common value of $f$ among the $k$ training examples nearest to $x_q$. If we choose $k = 1$, then the 1-NEAREST NEIGHBOR algorithm assigns to $\hat{f}(x_q)$ the value $f(x_i)$ where $x_i$ is the training instance nearest to $x_q$. For larger values of $k$, the algorithm assigns the most common value among the $k$ nearest training examples.

Figure 8.1 illustrates the operation of the $k$-NEAREST NEIGHBOR algorithm for the case where the instances are points in a two-dimensional space and where the target function is boolean valued. The positive and negative training examples are shown by "+" and "−" respectively. A query point $x_q$ is shown as well. Note the 1-NEAREST NEIGHBOR algorithm classifies $x_q$ as a positive example in this figure, whereas the 5-NEAREST NEIGHBOR algorithm classifies it as a negative example.

What is the nature of the hypothesis space $H$ implicitly considered by the $k$-NEAREST NEIGHBOR algorithm? Note the $k$-NEAREST NEIGHBOR algorithm never forms an explicit general hypothesis $\hat{f}$ regarding the target function $f$. It simply computes the classification of each new query instance as needed. Nevertheless,

---

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance $x_q$ to be classified,
  - Let $x_1 \ldots x_k$ denote the $k$ instances from *training_examples* that are nearest to $x_q$
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

---

**TABLE 8.1**
The $k$-NEAREST NEIGHBOR algorithm for approximating a discrete-valued function $f : \Re^n \rightarrow V$.

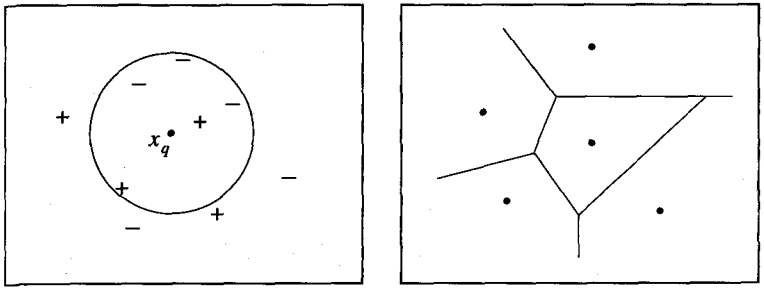


**FIGURE 8.1**
*k*-NEAREST NEIGHBOR. A set of positive and negative training examples is shown on the left, along with a query instance $x_q$ to be classified. The 1-NEAREST NEIGHBOR algorithm classifies $x_q$ positive, whereas 5-NEAREST NEIGHBOR classifies it as negative. On the right is the decision surface induced by the 1-NEAREST NEIGHBOR algorithm for a typical set of training examples. The convex polygon surrounding each training example indicates the region of instance space closest to that point (i.e., the instances for which the 1-NEAREST NEIGHBOR algorithm will assign the classification belonging to that training example).

we can still ask what the implicit general function is, or what classifications would be assigned if we were to hold the training examples constant and query the algorithm with every possible instance in $X$. The diagram on the right side of Figure 8.1 shows the shape of this decision surface induced by 1-NEAREST NEIGHBOR over the entire instance space. The decision surface is a combination of convex polyhedra surrounding each of the training examples. For every training example, the polyhedron indicates the set of query points whose classification will be completely determined by that training example. Query points outside the polyhedron are closer to some other training example. This kind of diagram is often called the *Voronoi diagram* of the set of training examples.

The *k*-NEAREST NEIGHBOR algorithm is easily adapted to approximating continuous-valued target functions. To accomplish this, we have the algorithm calculate the mean value of the $k$ nearest training examples rather than calculate their most common value. More precisely, to approximate a real-valued target function $f : \Re^n \rightarrow \Re$ we replace the final line of the above algorithm by the line

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k} \tag{8.1}$$

### 8.2.1 Distance-Weighted NEAREST NEIGHBOR Algorithm

One obvious refinement to the *k*-NEAREST NEIGHBOR algorithm is to weight the contribution of each of the $k$ neighbors according to their distance to the query point $x_q$, giving greater weight to closer neighbors. For example, in the algorithm of Table 8.1, which approximates discrete-valued target functions, we might weight the vote of each neighbor according to the inverse square of its distance from $x_q$.

This can be accomplished by replacing the final line of the algorithm by

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \delta(v, f(x_i)) \tag{8.2}$$

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2} \tag{8.3}$$

To accommodate the case where the query point $x_q$ exactly matches one of the training instances $x_i$ and the denominator $d(x_q, x_i)^2$ is therefore zero, we assign $\hat{f}(x_q)$ to be $f(x_i)$ in this case. If there are several such training examples, we assign the majority classification among them.

We can distance-weight the instances for real-valued target functions in a similar fashion, replacing the final line of the algorithm in this case by

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i} \tag{8.4}$$

where $w_i$ is as defined in Equation (8.3). Note the denominator in Equation (8.4) is a constant that normalizes the contributions of the various weights (e.g., it assures that if $f(x_i) = c$ for all training examples, then $\hat{f}(x_q) \leftarrow c$ as well).

Note all of the above variants of the $k$-NEAREST NEIGHBOR algorithm consider only the $k$ nearest neighbors to classify the query point. Once we add distance weighting, there is really no harm in allowing all training examples to have an influence on the classification of the $x_q$, because very distant examples will have very little effect on $\hat{f}(x_q)$. The only disadvantage of considering all examples is that our classifier will run more slowly. If all training examples are considered when classifying a new query instance, we call the algorithm a *global* method. If only the nearest training examples are considered, we call it a *local* method. When the rule in Equation (8.4) is applied as a global method, using all training examples, it is known as Shepard's method (Shepard 1968).

### 8.2.2 Remarks on $k$-NEAREST NEIGHBOR Algorithm

The distance-weighted $k$-NEAREST NEIGHBOR algorithm is a highly effective inductive inference method for many practical problems. It is robust to noisy training data and quite effective when it is provided a sufficiently large set of training data. Note that by taking the weighted average of the $k$ neighbors nearest to the query point, it can smooth out the impact of isolated noisy training examples.

What is the inductive bias of $k$-NEAREST NEIGHBOR? The basis for classifying new query points is easily understood based on the diagrams in Figure 8.1. The inductive bias corresponds to an assumption that the classification of an instance $x_q$ will be most similar to the classification of other instances that are nearby in Euclidean distance.

One practical issue in applying $k$-NEAREST NEIGHBOR algorithms is that the distance between instances is calculated based on *all* attributes of the instance

(i.e., on all axes in the Euclidean space containing the instances). This lies in contrast to methods such as rule and decision tree learning systems that select only a subset of the instance attributes when forming the hypothesis. To see the effect of this policy, consider applying $k$-NEAREST NEIGHBOR to a problem in which each instance is described by 20 attributes, but where only 2 of these attributes are relevant to determining the classification for the particular target function. In this case, instances that have identical values for the 2 relevant attributes may nevertheless be distant from one another in the 20-dimensional instance space. As a result, the similarity metric used by $k$-NEAREST NEIGHBOR—depending on all 20 attributes—will be misleading. The distance between neighbors will be dominated by the large number of irrelevant attributes. This difficulty, which arises when many irrelevant attributes are present, is sometimes referred to as the *curse of dimensionality*. Nearest-neighbor approaches are especially sensitive to this problem.

One interesting approach to overcoming this problem is to weight each attribute differently when calculating the distance between two instances. This corresponds to stretching the axes in the Euclidean space, shortening the axes that correspond to less relevant attributes, and lengthening the axes that correspond to more relevant attributes. The amount by which each axis should be stretched can be determined automatically using a cross-validation approach. To see how, first note that we wish to stretch (multiply) the $j$th axis by some factor $z_j$, where the values $z_1 \ldots z_n$ are chosen to minimize the true classification error of the learning algorithm. Second, note that this true error can be estimated using cross-validation. Hence, one algorithm is to select a random subset of the available data to use as training examples, then determine the values of $z_1 \ldots z_n$ that lead to the minimum error in classifying the remaining examples. By repeating this process multiple times the estimate for these weighting factors can be made more accurate. This process of stretching the axes in order to optimize the performance of $k$-NEAREST NEIGHBOR provides a mechanism for suppressing the impact of irrelevant attributes.

An even more drastic alternative is to completely eliminate the least relevant attributes from the instance space. This is equivalent to setting some of the $z_i$ scaling factors to zero. Moore and Lee (1994) discuss efficient cross-validation methods for selecting relevant subsets of the attributes for $k$-NEAREST NEIGHBOR algorithms. In particular, they explore methods based on leave-one-out cross-validation, in which the set of $m$ training instances is repeatedly divided into a training set of size $m-1$ and test set of size 1, in all possible ways. This leave-one-out approach is easily implemented in $k$-NEAREST NEIGHBOR algorithms because no additional training effort is required each time the training set is redefined. Note both of the above approaches can be seen as stretching each axis by some constant factor. Alternatively, we could stretch each axis by a value that varies over the instance space. However, as we increase the number of degrees of freedom available to the algorithm for redefining its distance metric in such a fashion, we also increase the risk of overfitting. Therefore, the approach of locally stretching the axes is much less common.

One additional practical issue in applying $k$-NEAREST NEIGHBOR is efficient memory indexing. Because this algorithm delays all processing until a new query is received, significant computation can be required to process each new query. Various methods have been developed for indexing the stored training examples so that the nearest neighbors can be identified more efficiently at some additional cost in memory. One such indexing method is the *kd*-tree (Bentley 1975; Friedman et al. 1977), in which instances are stored at the leaves of a tree, with nearby instances stored at the same or nearby nodes. The internal nodes of the tree sort the new query $x_q$ to the relevant leaf by testing selected attributes of $x_q$.

### 8.2.3 A Note on Terminology

Much of the literature on nearest-neighbor methods and weighted local regression uses a terminology that has arisen from the field of statistical pattern recognition. In reading that literature, it is useful to know the following terms:

- *Regression* means approximating a real-valued target function.
- *Residual* is the error $\hat{f}(x) - f(x)$ in approximating the target function.
- *Kernel function* is the function of distance that is used to determine the weight of each training example. In other words, the kernel function is the function $K$ such that $w_i = K(d(x_i, x_q))$.

## 8.3 LOCALLY WEIGHTED REGRESSION

The nearest-neighbor approaches described in the previous section can be thought of as approximating the target function $f(x)$ at the single query point $x = x_q$. Locally weighted regression is a generalization of this approach. It constructs an explicit approximation to $f$ over a local region surrounding $x_q$. Locally weighted regression uses nearby or distance-weighted training examples to form this local approximation to $f$. For example, we might approximate the target function in the neighborhood surrounding $x_q$ using a linear function, a quadratic function, a multilayer neural network, or some other functional form. The phrase "locally weighted regression" is called *local* because the function is approximated based only on data near the query point, *weighted* because the contribution of each training example is weighted by its distance from the query point, and *regression* because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.

Given a new query instance $x_q$, the general approach in locally weighted regression is to construct an approximation $\hat{f}$ that fits the training examples in the neighborhood surrounding $x_q$. This approximation is then used to calculate the value $\hat{f}(x_q)$, which is output as the estimated target value for the query instance. The description of $\hat{f}$ may then be deleted, because a different local approximation will be calculated for each distinct query instance.

### 8.3.1 Locally Weighted Linear Regression

Let us consider the case of locally weighted regression in which the target function $f$ is approximated near $x_q$ using a linear function of the form

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \cdots + w_n a_n(x)$$

As before, $a_i(x)$ denotes the value of the $i$th attribute of the instance $x$.

Recall that in Chapter 4 we discussed methods such as gradient descent to find the coefficients $w_0 \ldots w_n$ to minimize the error in fitting such linear functions to a given set of training examples. In that chapter we were interested in a global approximation to the target function. Therefore, we derived methods to choose weights that minimize the squared error summed over the set $D$ of training examples

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \tag{8.5}$$

which led us to the gradient descent training rule

$$\Delta w_j = \eta \sum_{x \in D} (f(x) - \hat{f}(x)) a_j(x) \tag{8.6}$$

where $\eta$ is a constant learning rate, and where the training rule has been re-expressed from the notation of Chapter 4 to fit our current notation (i.e., $t \rightarrow f(x)$, $o \rightarrow \hat{f}(x)$, and $x_j \rightarrow a_j(x)$).

How shall we modify this procedure to derive a local approximation rather than a global one? The simple way is to redefine the error criterion $E$ to emphasize fitting the local training examples. Three possible criteria are given below. Note we write the error $E(x_q)$ to emphasize the fact that now the error is being defined as a function of the query point $x_q$.

1. Minimize the squared error over just the $k$ nearest neighbors:

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2$$

2. Minimize the squared error over the entire set $D$ of training examples, while weighting the error of each training example by some decreasing function $K$ of its distance from $x_q$:

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \, K(d(x_q, x))$$

3. Combine 1 and 2:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \, K(d(x_q, x))$$

Criterion two is perhaps the most esthetically pleasing because it allows every training example to have an impact on the classification of $x_q$. However,

this approach requires computation that grows linearly with the number of training examples. Criterion three is a good approximation to criterion two and has the advantage that computational cost is independent of the total number of training examples; its cost depends only on the number $k$ of neighbors considered.

If we choose criterion three above and rederive the gradient descent rule using the same style of argument as in Chapter 4, we obtain the following training rule (see Exercise 8.1):

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest nbrs of } x_q} K(d(x_q, x))\ (f(x) - \hat{f}(x))\ a_j(x) \qquad (8.7)$$

Notice the only differences between this new rule and the rule given by Equation (8.6) are that the contribution of instance $x$ to the weight update is now multiplied by the distance penalty $K(d(x_q, x))$, and that the error is summed over only the $k$ nearest training examples. In fact, if we are fitting a linear function to a fixed set of training examples, then methods much more efficient than gradient descent are available to directly solve for the desired coefficients $w_0 \ldots w_n$. Atkeson et al. (1997a) and Bishop (1995) survey several such methods.

### 8.3.2 Remarks on Locally Weighted Regression

Above we considered using a linear function to approximate $f$ in the neighborhood of the query instance $x_q$. The literature on locally weighted regression contains a broad range of alternative methods for distance weighting the training examples, and a range of methods for locally approximating the target function. In most cases, the target function is approximated by a constant, linear, or quadratic function. More complex functional forms are not often found because (1) the cost of fitting more complex functions for each query instance is prohibitively high, and (2) these simple approximations model the target function quite well over a sufficiently small subregion of the instance space.

## 8.4 RADIAL BASIS FUNCTIONS

One approach to function approximation that is closely related to distance-weighted regression and also to artificial neural networks is learning with radial basis functions (Powell 1987; Broomhead and Lowe 1988; Moody and Darken 1989). In this approach, the learned hypothesis is a function of the form

$$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x)) \qquad (8.8)$$

where each $x_u$ is an instance from $X$ and where the kernel function $K_u(d(x_u, x))$ is defined so that it decreases as the distance $d(x_u, x)$ increases. Here $k$ is a user-provided constant that specifies the number of kernel functions to be included. Even though $\hat{f}(x)$ is a global approximation to $f(x)$, the contribution from each of the $K_u(d(x_u, x))$ terms is localized to a region nearby the point $x_u$. It is common

to choose each function $K_u(d(x_u, x))$ to be a Gaussian function (see Table 5.4) centered at the point $x_u$ with some variance $\sigma_u^2$.

$$K_u(d(x_u, x)) = e^{\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

We will restrict our discussion here to this common Gaussian kernel function. As shown by Hartman et al. (1990), the functional form of Equation (8.8) can approximate any function with arbitrarily small error, provided a sufficiently large number $k$ of such Gaussian kernels and provided the width $\sigma^2$ of each kernel can be separately specified.

The function given by Equation (8.8) can be viewed as describing a two-layer network where the first layer of units computes the values of the various $K_u(d(x_u, x))$ and where the second layer computes a linear combination of these first-layer unit values. An example radial basis function (RBF) network is illustrated in Figure 8.2.

Given a set of training examples of the target function, RBF networks are typically trained in a two-stage process. First, the number $k$ of hidden units is determined and each hidden unit $u$ is defined by choosing the values of $x_u$ and $\sigma_u^2$ that define its kernel function $K_u(d(x_u, x))$. Second, the weights $w_u$ are trained to maximize the fit of the network to the training data, using the global error criterion given by Equation (8.5). Because the kernel functions are held fixed during this second stage, the linear weight values $w_u$ can be trained very efficiently.

Several alternative methods have been proposed for choosing an appropriate number of hidden units or, equivalently, kernel functions. One approach is to allocate a Gaussian kernel function for each training example $\langle x_i, f(x_i)\rangle$, centering this Gaussian at the point $x_i$. Each of these kernels may be assigned the same width $\sigma^2$. Given this approach, the RBF network learns a global approximation to the target function in which each training example $\langle x_i, f(x_i)\rangle$ can influence the value of $\hat{f}$ only in the neighborhood of $x_i$. One advantage of this choice of kernel functions is that it allows the RBF network to fit the training data exactly. That is, for any set of $m$ training examples the weights $w_0 \ldots w_m$ for combining the $m$ Gaussian kernel functions can be set so that $\hat{f}(x_i) = f(x_i)$ for each training example $\langle x_i, f(x_i)\rangle$.

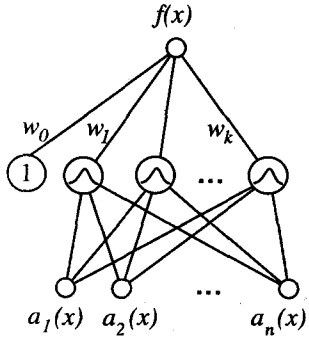


**FIGURE 8.2**
A radial basis function network. Each hidden unit produces an activation determined by a Gaussian function centered at some instance $x_u$. Therefore, its activation will be close to zero unless the input $x$ is near $x_u$. The output unit produces a linear combination of the hidden unit activations. Although the network shown here has just one output, multiple output units can also be included.

A second approach is to choose a set of kernel functions that is smaller than the number of training examples. This approach can be much more efficient than the first approach, especially when the number of training examples is large. The set of kernel functions may be distributed with centers spaced uniformly throughout the instance space $X$. Alternatively, we may wish to distribute the centers nonuniformly, especially if the instances themselves are found to be distributed nonuniformly over $X$. In this later case, we can pick kernel function centers by randomly selecting a subset of the training instances, thereby sampling the underlying distribution of instances. Alternatively, we may identify prototypical clusters of instances, then add a kernel function centered at each cluster. The placement of the kernel functions in this fashion can be accomplished using unsupervised clustering algorithms that fit the training instances (but not their target values) to a mixture of Gaussians. The EM algorithm discussed in Section 6.12.1 provides one algorithm for choosing the means of a mixture of $k$ Gaussians to best fit the observed instances. In the case of the EM algorithm, the means are chosen to maximize the probability of observing the instances $x_i$, given the $k$ estimated means. Note the target function value $f(x_i)$ of the instance does not enter into the calculation of kernel centers by unsupervised clustering methods. The only role of the target values $f(x_i)$ in this case is to determine the output layer weights $w_u$.

To summarize, radial basis function networks provide a global approximation to the target function, represented by a linear combination of many local kernel functions. The value for any given kernel function is non-negligible only when the input $x$ falls into the region defined by its particular center and width. Thus, the network can be viewed as a smooth linear combination of many local approximations to the target function. One key advantage to RBF networks is that they can be trained much more efficiently than feedforward networks trained with BACKPROPAGATION. This follows from the fact that the input layer and the output layer of an RBF are trained separately.

## 8.5 CASE-BASED REASONING

Instance-based methods such as $k$-NEAREST NEIGHBOR and locally weighted regression share three key properties. First, they are *lazy* learning methods in that they defer the decision of how to generalize beyond the training data until a new query instance is observed. Second, they classify new query instances by analyzing similar instances while ignoring instances that are very different from the query. Third, they represent instances as real-valued points in an $n$-dimensional Euclidean space. Case-based reasoning (CBR) is a learning paradigm based on the first two of these principles, but not the third. In CBR, instances are typically represented using more rich symbolic descriptions, and the methods used to retrieve similar instances are correspondingly more elaborate. CBR has been applied to problems such as conceptual design of mechanical devices based on a stored library of previous designs (Sycara et al. 1992), reasoning about new legal cases based on previous rulings (Ashley 1990), and solving planning and

scheduling problems by reusing and combining portions of previous solutions to similar problems (Veloso 1992).

Let us consider a prototypical example of a case-based reasoning system to ground our discussion. The CADET system (Sycara et al. 1992) employs case-based reasoning to assist in the conceptual design of simple mechanical devices such as water faucets. It uses a library containing approximately 75 previous designs and design fragments to suggest conceptual designs to meet the specifications of new design problems. Each instance stored in memory (e.g., a water pipe) is represented by describing both its structure and its qualitative function. New design problems are then presented by specifying the desired function and requesting the corresponding structure. This problem setting is illustrated in Figure 8.3. The top half of the figure shows the description of a typical stored case called a T-junction pipe. Its function is represented in terms of the qualitative relationships among the waterflow levels and temperatures at its inputs and outputs. In the functional description at its right, an arrow with a "+" label indicates that the variable at the arrowhead increases with the variable at its tail. For example, the output waterflow $Q_3$ increases with increasing input waterflow $Q_1$. Similarly,

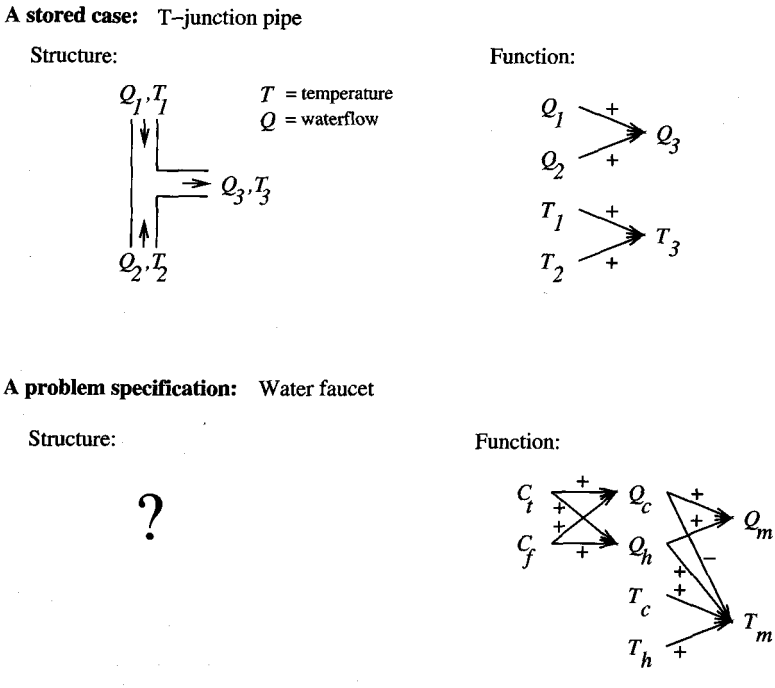


**FIGURE 8.3**
A stored case and a new problem. The top half of the figure describes a typical design fragment in the case library of CADET. The function is represented by the graph of qualitative dependencies among the T-junction variables (described in the text). The bottom half of the figure shows a typical design problem.

a "−" label indicates that the variable at the head decreases with the variable at the tail. The bottom half of this figure depicts a new design problem described by its desired function. This particular function describes the required behavior of one type of water faucet. Here $Q_c$ refers to the flow of cold water into the faucet, $Q_h$ to the input flow of hot water, and $Q_m$ to the single mixed flow out of the faucet. Similarly, $T_c$, $T_h$, and $T_m$ refer to the temperatures of the cold water, hot water, and mixed water respectively. The variable $C_t$ denotes the control signal for temperature that is input to the faucet, and $C_f$ denotes the control signal for waterflow. Note the description of the desired function specifies that these controls $C_t$ and $C_f$ are to influence the water flows $Q_c$ and $Q_h$, thereby indirectly influencing the faucet output flow $Q_m$ and temperature $T_m$.

Given this functional specification for the new design problem, CADET searches its library for stored cases whose functional descriptions match the design problem. If an exact match is found, indicating that some stored case implements exactly the desired function, then this case can be returned as a suggested solution to the design problem. If no exact match occurs, CADET may find cases that match various subgraphs of the desired functional specification. In Figure 8.3, for example, the T-junction function matches a subgraph of the water faucet function graph. More generally, CADET searches for subgraph isomorphisms between the two function graphs, so that parts of a case can be found to match parts of the design specification. Furthermore, the system may elaborate the original function specification graph in order to create functionally equivalent graphs that may match still more cases. It uses general knowledge about physical influences to create these elaborated function graphs. For example, it uses a rewrite rule that allows it to rewrite the influence

$$A \xrightarrow{+} B$$

as

$$A \xrightarrow{+} x \xrightarrow{+} B$$

This rewrite rule can be interpreted as stating that if $B$ must increase with $A$, then it is sufficient to find some other quantity $x$ such that $B$ increases with $x$, and $x$ increases with $A$. Here $x$ is a universally quantified variable whose value is bound when matching the function graph against the case library. In fact, the function graph for the faucet shown in Figure 8.3 is an elaboration of the original functional specification produced by applying such rewrite rules.

By retrieving multiple cases that match different subgraphs, the entire design can sometimes be pieced together. In general, the process of producing a final solution from multiple retrieved cases can be very complex. It may require designing portions of the system from first principles, in addition to merging retrieved portions from stored cases. It may also require backtracking on earlier choices of design subgoals and, therefore, rejecting cases that were previously retrieved. CADET has very limited capabilities for combining and adapting multiple retrieved cases to form the final design and relies heavily on the user for this adaptation stage of the process. As described by Sycara et al. (1992), CADET is

a research prototype system intended to explore the potential role of case-based reasoning in conceptual design. It does not have the range of analysis algorithms needed to refine these abstract conceptual designs into final designs.

It is instructive to examine the correspondence between the problem setting of CADET and the general setting for instance-based methods such as $k$-NEAREST NEIGHBOR. In CADET each stored training example describes a function graph along with the structure that implements it. New queries correspond to new function graphs. Thus, we can map the CADET problem into our standard notation by defining the space of instances $X$ to be the space of all function graphs. The target function $f$ maps function graphs to the structures that implement them. Each stored training example $\langle x, f(x)\rangle$ is a pair that describes some function graph $x$ and the structure $f(x)$ that implements $x$. The system must learn from the training example cases to output the structure $f(x_q)$ that successfully implements the input function graph query $x_q$.

The above sketch of the CADET system illustrates several generic properties of case-based reasoning systems that distinguish them from approaches such as $k$-NEAREST NEIGHBOR.

- Instances or cases may be represented by rich symbolic descriptions, such as the function graphs used in CADET. This may require a similarity metric different from Euclidean distance, such as the size of the largest shared subgraph between two function graphs.
- Multiple retrieved cases may be combined to form the solution to the new problem. This is similar to the $k$-NEAREST NEIGHBOR approach, in that multiple similar cases are used to construct a response for the new query. However, the process for combining these multiple retrieved cases can be very different, relying on knowledge-based reasoning rather than statistical methods.
- There may be a tight coupling between case retrieval, knowledge-based reasoning, and problem solving. One simple example of this is found in CADET, which uses generic knowledge about influences to rewrite function graphs during its attempt to find matching cases. Other systems have been developed that more fully integrate case-based reasoning into general search-based problem-solving systems. Two examples are ANAPRON (Golding and Rosenbloom 1991) and PRODIGY/ANALOGY (Veloso 1992).

To summarize, case-based reasoning is an instance-based learning method in which instances (cases) may be rich relational descriptions and in which the retrieval and combination of cases to solve the current query may rely on knowledge-based reasoning and search-intensive problem-solving methods. One current research issue in case-based reasoning is to develop improved methods for indexing cases. The central issue here is that syntactic similarity measures (e.g., subgraph isomorphism between function graphs) provide only an approximate indication of the relevance of a particular case to a particular problem. When the CBR system attempts to reuse the retrieved cases it may uncover difficulties that were not

captured by this syntactic similarity measure. For example, in CADET the multiple retrieved design fragments may turn out to be incompatible with one another, making it impossible to combine them into a consistent final design. When this occurs in general, the CBR system may backtrack and search for additional cases, adapt the existing cases, or resort to other problem-solving methods. Importantly, when such difficulties are detected they also provide training data for improving the similarity metric or, equivalently, the indexing structure for the case library. In particular, if a case is retrieved based on the similarity metric, but found to be irrelevant based on further analysis, then the similarity metric should be refined to reject this case for similar subsequent queries.

## 8.6 REMARKS ON LAZY AND EAGER LEARNING

In this chapter we considered three *lazy* learning methods: the $k$-NEAREST NEIGHBOR algorithm, locally weighted regression, and case-based reasoning. We call these methods lazy because they defer the decision of how to generalize beyond the training data until each new query instance is encountered. We also discussed one *eager* learning method: the method for learning radial basis function networks. We call this method eager because it generalizes beyond the training data before observing the new query, committing at training time to the network structure and weights that define its approximation to the target function. In this same sense, every other algorithm discussed elsewhere in this book (e.g., BACKPROPAGATION, C4.5) is an eager learning algorithm.

Are there important differences in what can be achieved by lazy versus eager learning? Let us distinguish between two kinds of differences: differences in computation time and differences in the classifications produced for new queries. There are obviously differences in computation time between eager and lazy methods. For example, lazy methods will generally require less computation during training, but more computation when they must predict the target value for a new query.

The more fundamental question is whether there are essential differences in the inductive bias that can be achieved by lazy versus eager methods. The key difference between lazy and eager methods in this regard is

- Lazy methods may consider the query instance $x_q$ when deciding how to generalize beyond the training data $D$.
- Eager methods cannot. By the time they observe the query instance $x_q$ they have already chosen their (global) approximation to the target function.

Does this distinction affect the generalization accuracy of the learner? It does if we require that the lazy and eager learner employ the same hypothesis space $H$. To illustrate, consider the hypothesis space consisting of linear functions. The locally weighted linear regression algorithm discussed earlier is a lazy learning method based on this hypothesis space. For each new query $x_q$ it generalizes from the training data by choosing a new hypothesis based on the training examples near $x_q$. In contrast, an eager learner that uses the same hypothesis space of linear functions

must choose its approximation before the queries are observed. The eager learner must therefore commit to a single linear function hypothesis that covers the entire instance space and all future queries. The lazy method effectively uses a richer hypothesis space because it uses many different local linear functions to form its implicit global approximation to the target function. Note this same situation holds for other learners and hypothesis spaces as well. A lazy version of BACKPROPAGATION, for example, could learn a different neural network for each distinct query point, compared to the eager version of BACKPROPAGATION discussed in Chapter 4.

The key point in the above paragraph is that a lazy learner has the option of (implicitly) representing the target function by a combination of many local approximations, whereas an eager learner must commit at training time to a single global approximation. The distinction between eager and lazy learning is thus related to the distinction between global and local approximations to the target function.

Can we create eager methods that use multiple local approximations to achieve the same effects as lazy local methods? Radial basis function networks can be seen as one attempt to achieve this. The RBF learning methods we discussed are eager methods that commit to a global approximation to the target function at training time. However, an RBF network represents this global function as a linear combination of multiple local kernel functions. Nevertheless, because RBF learning methods must commit to the hypothesis before the query point is known, the local approximations they create are not specifically targeted to the query point to the same degree as in a lazy learning method. Instead, RBF networks are built eagerly from local approximations centered around the training examples, or around clusters of training examples, but not around the unknown future query points.

To summarize, lazy methods have the option of selecting a different hypothesis or local approximation to the target function for each query instance. Eager methods using the same hypothesis space are more restricted because they must commit to a single hypothesis that covers the entire instance space. Eager methods can, of course, employ hypothesis spaces that combine multiple local approximations, as in RBF networks. However, even these combined local approximations do not give eager methods the full ability of lazy methods to customize to unknown future query instances.

## 8.7 SUMMARY AND FURTHER READING

The main points of this chapter include:

- Instance-based learning methods differ from other approaches to function approximation because they delay processing of training examples until they must label a new query instance. As a result, they need not form an explicit hypothesis of the entire target function over the entire instance space, independent of the query instance. Instead, they may form a different local approximation to the target function for each query instance.

- Advantages of instance-based methods include the ability to model complex target functions by a collection of less complex local approximations and the fact that information present in the training examples is never lost (because the examples themselves are stored explicitly). The main practical difficulties include efficiency of labeling new instances (all processing is done at query time rather than in advance), difficulties in determining an appropriate distance metric for retrieving "related" instances (especially when examples are represented by complex symbolic descriptions), and the negative impact of irrelevant features on the distance metric.
- $k$-NEAREST NEIGHBOR is an instance-based algorithm for approximating real-valued or discrete-valued target functions, assuming instances correspond to points in an $n$-dimensional Euclidean space. The target function value for a new query is estimated from the known values of the $k$ nearest training examples.
- Locally weighted regression methods are a generalization of $k$-NEAREST NEIGHBOR in which an explicit local approximation to the target function is constructed for each query instance. The local approximation to the target function may be based on a variety of functional forms such as constant, linear, or quadratic functions or on spatially localized kernel functions.
- Radial basis function (RBF) networks are a type of artificial neural network constructed from spatially localized kernel functions. These can be seen as a blend of instance-based approaches (spatially localized influence of each kernel function) and neural network approaches (a global approximation to the target function is formed at training time rather than a local approximation at query time). Radial basis function networks have been used successfully in applications such as interpreting visual scenes, in which the assumption of spatially local influences is well-justified.
- Case-based reasoning is an instance-based approach in which instances are represented by complex logical descriptions rather than points in a Euclidean space. Given these complex symbolic descriptions of instances, a rich variety of methods have been proposed for mapping from the training examples to target function values for new instances. Case-based reasoning methods have been used in applications such as modeling legal reasoning and for guiding searches in complex manufacturing and transportation planning problems.

The $k$-NEAREST NEIGHBOR algorithm is one of the most thoroughly analyzed algorithms in machine learning, due in part to its age and in part to its simplicity. Cover and Hart (1967) present early theoretical results, and Duda and Hart (1973) provide a good overview. Bishop (1995) provides a discussion of $k$-NEAREST NEIGHBOR and its relation to estimating probability densities. An excellent current survey of methods for locally weighted regression is given by Atkeson et al. (1997). The application of these methods to robot control is surveyed by Atkeson et al. (1997b).

A thorough discussion of radial basis functions is provided by Bishop (1995). Other treatments are given by Powell (1987) and Poggio and Girosi (1990). See Section 6.12 of this book for a discussion of the EM algorithm and its application to selecting the means of a mixture of Gaussians.

Kolodner (1993) provides a general introduction to case-based reasoning. Other general surveys and collections describing recent research are given by Aamodt et al. (1994), Aha et al. (1991), Haton et al. (1995), Riesbeck and Schank (1989), Schank et al. (1994), Veloso and Aamodt (1995), Watson (1995), and Wess et al. (1994).

## EXERCISES

**8.1.** Derive the gradient descent rule for a distance-weighted local linear approximation to the target function, given by Equation (8.1).

**8.2.** Consider the following alternative method for accounting for distance in weighted local regression. Create a virtual set of training examples $D'$ as follows: For each training example $\langle x, f(x)\rangle$ in the original data set $D$, create some (possibly fractional) number of copies of $\langle x, f(x)\rangle$ in $D'$, where the number of copies is $K(d(x_q, x))$. Now train a linear approximation to minimize the error criterion

$$E_4 \equiv \frac{1}{2}\sum_{x \in D'}(f(x) - \hat{f}(x))^2$$

The idea here is to make more copies of training examples that are near the query instance, and fewer of those that are distant. Derive the gradient descent rule for this criterion. Express the rule in the form of a sum over members of $D$ rather than $D'$, and compare it with the rules given by Equations (8.6) and (8.7).

**8.3.** Suggest a lazy version of the eager decision tree learning algorithm ID3 (see Chapter 3). What are the advantages and disadvantages of your lazy algorithm compared to the original eager algorithm?

## REFERENCES

Aamodt, A., & Plazas, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1), 39–52.

Aha, D., & Kibler, D. (1989). Noise-tolerant instance-based learning algorithms. *Proceedings of the IJCAI-89* (794–799).

Aha, D., Kibler, D., & Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, 6, 37–66.

Ashley, K. D. (1990). *Modeling legal argument: Reasoning with cases and hypotheticals*. Cambridge, MA: MIT Press.

Atkeson, C. G., Schaal, S. A., & Moore, A. W. (1997a). Locally weighted learning. *AI Review*, (to appear).

Atkeson, C. G., Moore, A. W., & Schaal, S. A. (1997b). Locally weighted learning for control. *AI Review*, (to appear).

Bareiss, E. R., Porter, B., & Weir, C. C. (1988). PROTOS: An exemplar-based learning apprentice. *International Journal of Man-Machine Studies*, 29, 549–561.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9), 509–517.

Bishop, C. M. (1995). *Neural networks for pattern recognition.* Oxford, England: Oxford University Press.

Bisio, R., & Malabocchia, F. (1995). Cost estimation of software projects through case-based reasoning. In M. Veloso and A. Aamodt (Eds.), *Lecture Notes in Artificial Intelligence* (pp. 11–22). Berlin: Springer-Verlag.

Broomhead, D. S., & Lowe, D. (1988). Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2, 321–355.

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13, 21–27.

Duda, R., & Hart, P. (1973). *Pattern classification and scene analysis.* New York: John Wiley & Sons.

Franke, R. (1982). Scattered data interpolation: Tests of some methods. *Mathematics of Computation*, 38, 181–200.

Friedman, J., Bentley, J., & Finkel, R. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3), 209–226.

Golding, A., & Rosenbloom, P. (1991). Improving rule-based systems through case-based reasoning. *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 22–27). Cambridge: AAAI Press/The MIT Press.

Hartman, E. J., Keller, J. D., & Kowalski, J. M. (1990). Layered neural networks with Gaussian hidden units as universal approximations. *Neural Computation*, 2(2), 210–215.

Haton, J.-P., Keane, M., & Manago, M. (Eds.). (1995). *Advances in case-based reasoning: Second European workshop.* Berlin: Springer-Verlag.

Kolodner, J. L. (1993). *Case-Based Reasoning.* San Francisco: Morgan Kaufmann.

Moody, J. E., & Darken, C. J. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2), 281–294.

Moore, A. W., & Lee, M. S. (1994). Efficient algorithms for minimizing cross validation error. *Proceedings of the 11th International Conference on Machine Learning.* San Francisco: Morgan Kaufmann.

Poggio, T., & Girosi, F. (1990). Networks for approximation and learning. *Proceedings of the IEEE*, 78(9), 1481–1497.

Powell, M. J. D. (1987). Radial basis functions for multivariable interpolation: A review. In Mason, J., & Cox, M. (Eds.). *Algorithms for approximation* (pp. 143–167). Oxford: Clarendon Press.

Riesbeck, C., & Schank, R. (1989). *Inside case-based reasoning.* Hillsdale, NJ: Lawrence Erlbaum.

Schank, R. (1982). *Dynamic Memory.* Cambridge, England: Cambridge University Press.

Schank, R., Riesbeck, C., & Kass, A. (1994). *Inside case-based explanation.* Hillsdale, NJ: Lawrence Erlbaum.

Shepard, D. (1968). A two-dimensional interpolation function for irregularly spaced data. *Proceedings of the 23rd National Conference of the ACM* (pp. 517–523).

Stanfill, C., & Waltz, D. (1986). Toward memory-based reasoning. *Communications of the ACM*, 29(12), 1213–1228.

Sycara, K., Guttal, R., Koning, J., Narasimhan, S., & Navinchandra, D. (1992). CADET: A case-based synthesis tool for engineering design. *International Journal of Expert Systems*, 4(2), 157–188.

Veloso, M. M. (1992). *Planning and learning by analogical reasoning.* Berlin: Springer-Verlag.

Veloso, M. M., & Aamodt, A. (Eds.). (1995). *Case-based reasoning research and development.* Lecture Notes in Artificial Intelligence. Berlin: Springer-Verlag.

Watson, I. (Ed.). (1995). *Progress in case-based reasoning: First United Kingdom workshop.* Berlin: Springer-Verlag.

Wess, S., Althoff, K., & Richter, M. (Eds.). (1994). *Topics in case-based reasoning.* Berlin: Springer-Verlag.

# CHAPTER 9

# GENETIC ALGORITHMS

Genetic algorithms provide an approach to learning that is based loosely on simulated evolution. Hypotheses are often described by bit strings whose interpretation depends on the application, though hypotheses may also be described by symbolic expressions or even computer programs. The search for an appropriate hypothesis begins with a population, or collection, of initial hypotheses. Members of the current population give rise to the next generation population by means of operations such as random mutation and crossover, which are patterned after processes in biological evolution. At each step, the hypotheses in the current population are evaluated relative to a given measure of fitness, with the most fit hypotheses selected probabilistically as seeds for producing the next generation. Genetic algorithms have been applied successfully to a variety of learning tasks and to other optimization problems. For example, they have been used to learn collections of rules for robot control and to optimize the topology and learning parameters for artificial neural networks. This chapter covers both genetic algorithms, in which hypotheses are typically described by bit strings, and genetic programming, in which hypotheses are described by computer programs.

## 9.1 MOTIVATION

Genetic algorithms (GAs) provide a learning method motivated by an analogy to biological evolution. Rather than search from general-to-specific hypotheses, or from simple-to-complex, GAs generate successor hypotheses by repeatedly mutating and recombining parts of the best currently known hypotheses. At each step,