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 rth 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 $f : \Re^n \to V$, 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*
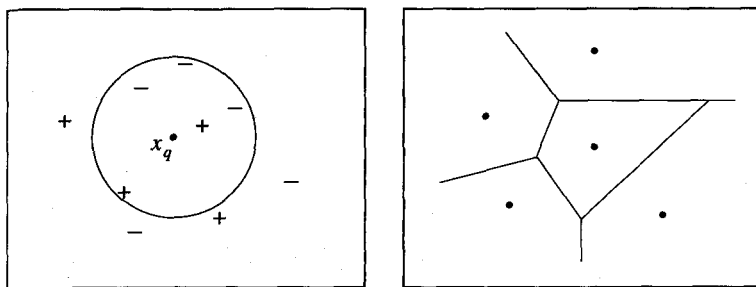
Classification algorithm:
- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

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

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 $f : \Re^n \to V$.

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 $f : \Re^n \to \Re$ 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

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

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 jth 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.