Search and learning Strategies

- T. Mitchell, Machine Learning, Chapter 8: *Instance-based Learning*, pp. 230-236, McGraw-Hill, 1997.
- M.J. Berry and G. Linoff, Data Mining Techniques, Chapter 10: *Automatic Cluster Detection*, pp. 187-215, John Wiley & Sons, 1997.

CHAPTER 8

INSTANCE-BASED LEARNING

Machine Learning
Tom Mitchell
McGraw-Hill

Study
Ready your textbook

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 $f : \Re^n \rightarrow V$, 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 $f : \Re^n \rightarrow V$.

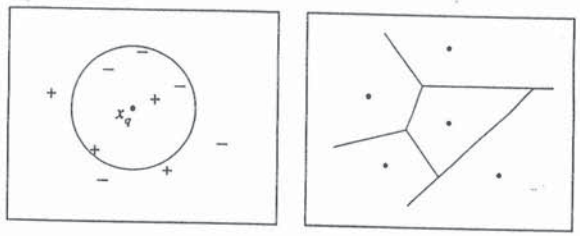

FIGURE 8.1
k-NEAREST NEIGHBOR. A set of positive and negative training examples is shown on the left, along with a query instance x_q to be classified. The 1-NEAREST NEIGHBOR algorithm classifies x_q positive, whereas 5-NEAREST NEIGHBOR classifies it as negative. On the right is the decision surface 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 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.

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 ith 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 η 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,

prediction, this phase is always rather expensive because finding the nearest neighbors involves applying the distance function to all the fields in the record and all the records in the training set. By contrast, decision trees and neural networks incorporate the training set into their models, then discard the training set.

Large Amount of Storage for Training Set

The training set used by MBR is the model, and the larger the training set the better the results. Although there are some techniques for reducing the number of records in the training set, the remaining records must still be represented. By contrast, the size of a neural network model depends only on the topology of the network and has no dependency on the size of the training set.

Dependence on Distance Function and Combination Function

The results from MBR do depend on the particular choice of distance function, combination function, and k, the number of neighbors chosen. Different choices can affect the results. Fortunately, common choices work pretty well. It is fairly easy to use a test set to find the best choice of k. Only if the results for all values of k are disappointing should you consider changing the distance and combination functions.

WHEN TO APPLY MEMORY-BASED REASONING

Memory-based reasoning is a directed data mining technique useful for both classification and prediction. In comparison with other techniques, it works well when the patterns in the data are likely to be highly local, so global rules and global functions do not make sense. For example, there may be many reasons why customers stop buying from a particular catalog—new shops open in their neighborhood, they are inundated by other catalogs, they lose their job, get a big promotion, and so forth. This is a good example of where MBR would be very applicable. On the other hand, there are fewer reasons why customers regularly make purchases in December—probably for giving gifts—so MBR is probably less useful to predict which customer needs a holiday catalog.

In short, MBR is a powerful technique that incorporates local information for classification and prediction purposes. The more complicated the data, the more likely it is that local patterns dominate the patterns, making MBR useful in many different circumstances.

Data Mining Techniques
M.J.A. Berry and G. Linoff
John Wiley & Sons, Inc.

Study voor opdracht-tentamen

10

Automatic Cluster Detection

We are always being told to "look at the big picture." But the fact is, sometimes the big picture is too confusing to be understood. A large database may contain so many variables, so many dimensions, and so much complex structure that even the best-directed data mining techniques are unable to coax meaningful patterns from it. In many cases, the problem is not that there are no patterns to be found, but that there are too many. When mining such a database for the answer to some specific question, we often find nothing but noise. Competing explanations cancel each other out.

When human beings try to make sense of complex questions, our natural tendency is to break the subject into smaller pieces, each of which can be explained more simply. If someone were to ask you to describe the color of trees in the forest, your answer would probably make distinctions between deciduous trees, conifers, and other evergreens, and between winter, spring, summer, and fall. You know enough about woodland flora to predict that, of all the hundreds of variables associated with the forest, season and foliage type, rather than say altitude and soil acidity, are the best discriminators to use for forming clusters of trees that follow similar coloration rules.

In marketing terms, subdividing the population according to variables already known to be good discriminators is called "segmentation." But in many cases, although we may suspect that a very noisy dataset is actually composed of a number of better behaved clusters,

we have no idea how to define them. That's where techniques for automatic cluster detection come in—when you can't see the forest without knowing a bit more about the trees.

SEARCHING FOR ISLANDS OF SIMPLICITY

Clustering is one of the few data mining activities that can properly be described as undirected knowledge discovery or unsupervised learning. For most data mining tasks, we start out with a preclassified training set and attempt to develop a model capable of predicting how a new record will be classified. In clustering, there is no preclassified data and no distinction between independent and dependent variables. Instead, we are searching for groups of records—the clusters—that are similar to one another, in the expectation that similar records represent similar customers or suppliers or products that will behave in similar ways.

Automatic cluster detection is rarely used in isolation because finding clusters is not an end in itself. Once clusters have been detected, other methods must be applied in order to figure out what the clusters mean. When clustering is successful, the results can be dramatic: One famous early application of cluster detection led to our current understanding of stellar evolution.

Star Light, Star Bright

Early in this century, astronomers trying to understand the relationship between the luminosity (brightness) of stars and their temperatures, made scatter plots like the one in Figure 10.1. The vertical scale measures luminosity in multiples of the brightness of our own sun. The horizontal scale measures surface temperature in degrees Kelvin (degrees centigrade above absolute 0, the theoretical coldest possible temperature where molecular motion ceases).

As you can see, the stars plotted by astronomers, Hertzsprung and Russell, fall into three clusters. We now understand that these three clusters represent stars in very different phases in the stellar life cycle. The relationship between luminosity and temperature is consistent within each cluster, but the relationship is different in each cluster because a fundamentally different process is generating the heat and light. The 80 percent of stars that fall on the main sequence are generating energy by converting hydrogen to helium through nuclear fusion. This is how all stars spend most of their life. But after 10 billion years or so, the hydrogen gets used up. Depending on the star's mass, it then begins fusing helium or the fusion stops. In the latter case, the core of the star begins to collapse, generating a great deal of heat in the process. At the same time, the outer layer of gasses expands away from the core. A red giant is formed. Eventually, the outer layer of gasses is stripped away and the remaining core begins to cool. The star is now a white dwarf.

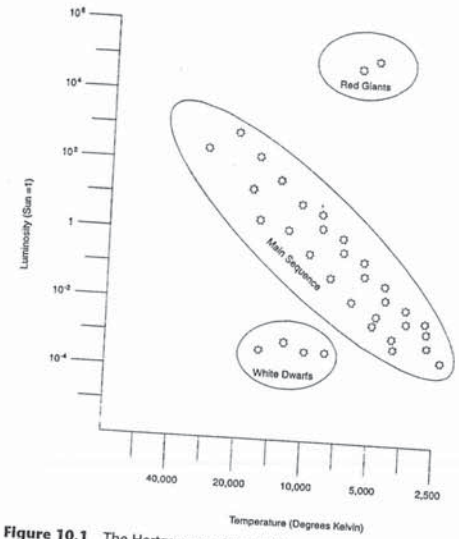

Figure 10.1 The Hertzsprung-Russell diagram clusters stars by temperature and luminosity.

A recent query of the Alta Vista web index using the search terms "HR diagram" and "main sequence" returned many pages of links to current astronomical research based on cluster detection of this kind. This simple, two-variable cluster diagram is being used today to hunt for new kinds of stars like "brown dwarfs" and to understand pre-main sequence stellar evolution.

Fitting the Troops

We chose the Hertzsprung-Russell diagram as our introductory example of clustering because with only two variables, it is easy to spot the clusters visually. Even in three dimensions, it is easy to pick out clusters by eye from a scatter plot cube. If all problems had so few dimensions, there would be no need for automatic cluster detection algorithms. As the number of dimensions (independent variables) increases, our ability to visualize clusters and our intuition about the distance between two points quickly break down.

When we speak of a problem as having many dimensions, we are making a geometric analogy. We consider each of the things that must be measured independently in order to describe something to be a *dimension*. In other words, if there are N variables, we imagine a space in which the value of each variable represents a distance along the corresponding axis in an N-dimensional space. A single record containing a value for each of the N variables, can be thought of as the vector that defines a particular point in that space. Figure 10.2 plots the height and weight of a group of teenagers as points on a graph. Notice the clustering of boys and girls.

The chart in Figure 10.2 begins to give a rough idea of people's shapes. But if we wanted to fit them for clothes, we would need many more measurements! The U.S. army recently commissioned a study on how to redesign the uniforms of female soldiers. The army's goal is to reduce the number of different uniform sizes that have to be kept in inventory while still providing each soldier with well-fitting khakis.

As anyone who has ever shopped for women's clothing is aware, there is already a surfeit of classification systems (odd sizes, even sizes, junior, petite, etc.) for categorizing garments by size, but none of these systems was designed with the needs of the U.S. military in mind. Susan Ashdown and Beatrix Paal, researchers at Cornell University, went back to the basics; they designed a new set of sizes based on the actual shapes of women in the army.

Unlike the traditional clothing size systems, the one Ashdown and Paal came up with is not an ordered set of graduated sizes where all dimensions increase together. Instead, they came up with sizes that fit particular body types. So, one size might be for short-legged, small-waisted, large-busted women with long torsos, average arms, broad shoulders, and skinny necks while other sizes fit other constellations of measurements.

The database they mined contained more than 100 measurements for each of nearly 3,000 women. The clustering technique employed in

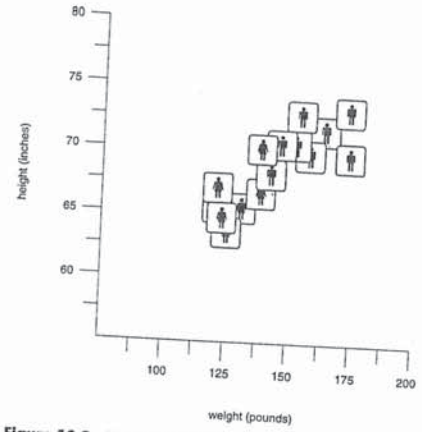

Figure 10.2 Height and weight of a group of teenagers.

this case was the K-means algorithm. In this approach, the first step is to choose the number of clusters (uniform sizes, in this case) that you want to form. That number is the *K* in K-means. Next, K "seeds" are chosen to be the initial guess at the centroids of the clusters. Each seed is just a particular combination of values for each measurement. Each seed might record the actual measurements of one of the women in the sample, but that is not a requirement.

Next, each record in the database is given a preliminary cluster assignment based on the seed to which it is closest. Then the centroids (or *means*) of the new clusters are calculated and the whole process starts over with the new centroids taking on the role of the seeds. Since the new centroids will not be in the same place as the original seeds, some of the records will be moved from the first cluster to which they were assigned to another one. (Actually, it is the cluster boundaries that move, not the points described by the records.) After a few iterations, this motion stops and the centroid of each cluster contains the measurements that define one of the new uniform sizes.

Spotting the Entrepreneurs

A third application of automatic cluster detection is described in Chapter 3. In that example, a bank used an automatic cluster detection technique to find clusters of similar customers in its customer information warehouse.

Although the clothing size example was undirected data mining in the sense that there were no pre-defined size categories to be found, the bank's quest was even less-directed. In the former case, the number of clusters to be found was imposed externally by the army which only wants to deal with a certain small number of uniform sizes. Furthermore, there could be no doubt about the meaning of the clusters once found. Since all the variables are body measurements, they are all in comparable units and each cluster clearly represents a certain body type.

The bank, on the other hand, has variables that measure many different things in many different units. There is no clear right way to compare outstanding balances with customer tenure or home zip code. The clusters found will depend greatly on the way these dissimilar values are scaled and weighted. With the bank's data, there is no obvious way to chose a value for K, the number of clusters to be formed. Worse, there is no readily available interpretation of any clusters that are found.

In fact, the bank found 14 clusters and only came up with a useful interpretation of one of them. But, that one cluster was so useful that nobody minded that they couldn't make sense of the other 13. The useful cluster was very rich in people who had both personal and business accounts with the bank and with people rated likely to respond to a home equity loan offer. This combination of traits led the bank to a new marketing premise—that people take out home equity loans in order to start a small business.

THE K-MEANS METHOD

The K-means method of cluster detection is the most commonly used in practice. It has many variations, but the form described here was first published by J. B. MacQueen in 1967. For ease of drawing, we illustrate the process using two-dimensional diagrams, but bear in mind that in practice we will usually be working in a space of many more dimensions. That means that instead of points described by a two-element vector (x_1,x_2), we work with points described by an n-element vector $(x_1,x_2, \ldots, x_n)$. The procedure itself is unchanged.

In the first step, we select K data points to be the seeds. MacQueen's algorithm simply takes the first K records. In cases where the records have some meaningful order, it may be desirable to choose widely spaced records instead. Each of the seeds is an embryonic cluster with only one element. In this example, we use outside information about the data to set the number of clusters to 3.

In the second step, we assign each record to the cluster whose centroid is nearest. In Figure 10.3 we have done the first two steps. Drawing the boundaries between the clusters is easy if you recall from high school geometry that given two points, X and Y, all points that are equidistant from X and Y fall along a line that is half way along the line segment that joins X and Y and perpendicular to it. In Figure 10.3, the initial seeds are joined by dashed lines and the cluster boundaries constructed from them are solid lines. Of course, in three dimensions, these boundaries would be planes and in N dimensions they would be hyperplanes of dimension N – 1.

As we continue to work through the K-means algorithm, pay particular attention to the fate of the point with the box drawn around it.

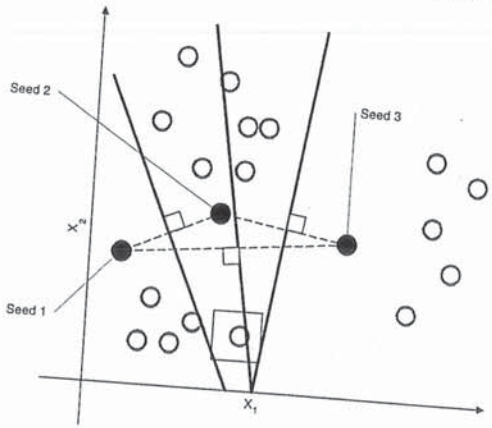

Figure 10.3 The initial seeds determine the initial cluster boundaries.

On the basis of the initial seeds, it is assigned to the cluster controlled by seed number 2 because it is closer to that seed than to either of the others.

At this point, every point has been assigned to one or another of the three clusters centered about the original seeds. The next step is to calculate the centroids of the new clusters. This is simply a matter of averaging the positions of each point in the cluster along each dimension. If there are 200 records assigned to a cluster and we are clustering based on four fields from those records, then geometrically we have 200 points in a 4-dimensional space. The location of each point is described by a vector of the values of the four fields. The vectors have the form (X_1,X_2,X_3,X_4). The value of X_1 for the new centroid is the mean of all 200 X_1s and similarly for X_2, X_3, and X_4.

In Figure 10.4, the new centroids are marked with a cross. The arrows show the motion from the position of the original seeds to the new centroids of the clusters formed from those seeds.

Once the new clusters have been found, each point is once again

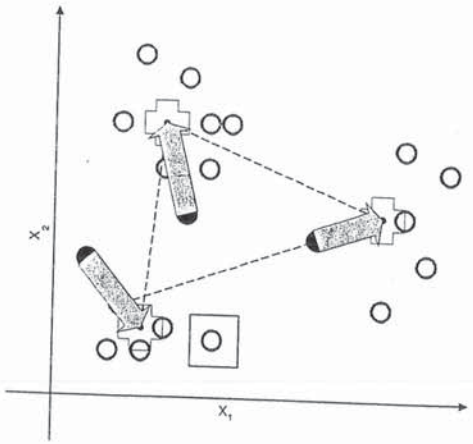

Figure 10.4 Calculating the centroids of the new clusters.

assigned to the cluster with the closest centroid. Figure 10.5 shows the new cluster boundaries—formed, as before, by drawing lines equidistant between each pair of centroids. Notice that the point with the box around it, which was originally assigned to cluster number 2, has now been assigned to cluster number 1. The process of assigning points to cluster and then re-calculating centroids continues until the cluster boundaries stop changing.

Similarity, Association, and Distance

After reading the preceding description of the K-means algorithm, we hope you agree that once the records in a database have been mapped to points in space, automatic cluster detection is really quite simple—a little geometry, some vector means, *et voilà*!

The problem, of course, is that the databases we encounter in marketing, sales, and customer support are not about points in space. They are about purchases, phone calls, airplane trips, car registrations,

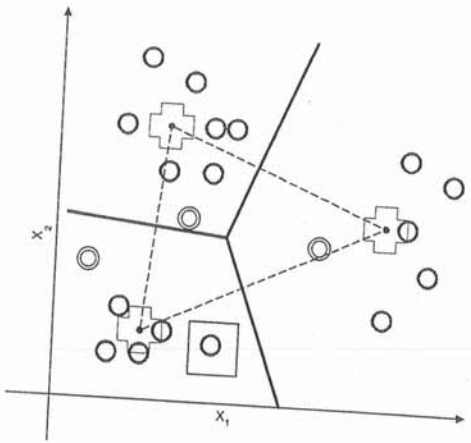

Figure 10.5 At each iteration, all cluster assignments are re-evaluated.

and a thousand other things that have no obvious connection to the dots in a cluster diagram.

When we speak of clustering records of this sort, we have an intuitive notion that members of a cluster have some kind of *natural association*; that they are more *similar* to each other than to records in another cluster. Since it is difficult to convey intuitive notions to a computer, we must translate the vague concept of association into some sort of numeric measure of the degree of similarity. The most common method, but by no means the only one, is to translate all fields into numeric values so that the records may be treated as points in space. Then, if two points are close in the geometric sense, we assume that they represent similar records in the database. There are two main problems with this approach:

1. Many variable types, including all categorical variables and many numeric variables such as rankings, do not have the right behavior to properly be treated as components of a position vector.
2. In geometry, the contributions of each dimension are of equal importance, but in our databases, a small change in one field may be much more important than a large change in another field.

A Variety of Variables

Variables can be categorized in various ways—by mathematical properties (continuous, discrete), by storage type (character, integer, floating point), and by other properties (quantitative, qualitative). For this discussion, however, the most important classification is how much the variable can tell us about its placement along the axis that corresponds to it in our geometric model. For this purpose, we can divide variables into four classes, listed here in increasing order of suitability for the geometric model.

Categories
Ranks
Intervals
True measures

Categorical variables only tell us to which of several unordered categories a thing belongs. We can say that this ice cream is pistachio while that one is mint-cookieo, but we cannot say that one is greater than the other or judge which one is closer to black cherry. In mathematical terms, we can tell that $X \neq Y$, but not whether $X > Y$ or $Y < X$.

Ranks allow us to put things in order, but don't tell us how much bigger one thing is than another. The valedictorian has better grades than the salutatorian, but we don't know by how much. If X, Y, and Z are ranked 1, 2, and 3, we know that $X > Y > Z$, but not whether $(X - Y) > (Y - Z)$.

Intervals allow us to measure the distance between two observations. If we are told that it is 56° in San Francisco and 78° in San Jose, we know that it is 22 degrees warmer at one end of the bay than the other.

True measures are interval variables that measure from a meaningful zero point. This trait is important because it means that the ratio of two values of the variable is meaningful. The Fahrenheit temperature scale used in the United States and the Celsius scale used in most of the rest of the world do not have this property. In neither system does it make sense to say that a 30° day is twice as warm as a 15° day. Similarly, a size 12 dress is not twice as large as a size 6 and gypsum is not twice as hard as talc though they are 2 and 1 on the hardness scale. It does make perfect sense, however, to say that a 50-year-old is twice as old as a 25-year-old or that a 10-pound bag of sugar is twice as heavy as a 5-pound one. Age, weight, length, and volume are examples of true measures.

Geometric distance metrics are well-defined for interval variables and true measures. In order to use categorical variables and rankings, it is necessary to transform them into interval variables. Unfortunately, these transformations add spurious information. If we number ice cream flavors 1 through 28, it will appear that flavors 5 and 6 are closely related while flavors 1 and 28 are far apart. The inverse problem arises when we transform interval variables and true measures into ranks or categories. As we go from age (true measure) to seniority (position on a list) to broad categories like "veteran" and "new hire," we lose information at each step.

There is further discussion of these conversion issues as they apply to distance metrics in Chapter 9.

Formal Measures of Association

There are dozens if not hundreds of published techniques for measuring the similarity of two records. Some have been developed for specialized applications such as comparing passages of text. Others are designed especially for use with certain types of data such as binary variables or categorical variables. Of the three we present here, the first two are suitable for use with interval variables and true measures while the third is suitable for categorical variables.

The Distance between Two Points

Each field in a record becomes one element in a vector describing a point in space. The distance between two points is used as the measure of association. If two points are close in distance, the corresponding records are considered similar. There are actually a number of metrics that can be used to measure the distance between two points (see aside), but the most common one is the Euclidian distance we all learned in high school. To find the Euclidian distance between X and Y, we first find the differences between the corresponding elements of X and Y (the distance along each axis) and square them. The distance is the square root of the sum of the squared differences.

Distance Metrics

Any function that takes two points and produces a single number describing a relationship between them is a candidate measure of association, but to be a true distance metric, it must meet the following criteria:

- `Distance(X,Y)` = 0 if and only if X = Y
- `Distance(X,Y)` ≥ 0 for all X and all Y
- `Distance(X,Y)` = `Distance(Y,X)`
- `Distance(X,Y)` ≤ `Distance(X,Z)` + `Distance(Z,Y)`

The Angle between Two Vectors

Sometimes we would like to consider two records to be closely associated because of similarities in the way the fields *within* each record are related. We would like to cluster minnows with sardines, cod, and tuna, while clustering kittens with cougars, lions, and tigers even though in a database of body-part lengths, the sardine is closer to the kitten than it is to the tuna.

The solution is to use a different geometric interpretation of the same data. Instead of thinking of X and Y as points in space and measuring the distance between them, we think of them as *vectors* and measure the *angle* between them. In this context, a vector is the line segment connecting the origin of our coordinate system to the point described by the vector values. A vector has both magnitude (the distance from the origin to the point) and direction. For our purposes, it is the direction that matters.

If we take the values for length of whiskers, length of tail, overall body length, length of teeth, and length of claws for a lion and a house cat and plot them as single points, they will be very far apart. But if the ratios of lengths of these body parts to one another are similar in the two species, than the vectors will be nearly parallel.

The angle between vectors provides a measure of association that is not influenced by differences in magnitude between the two things being compared (see Figure 10.6). Actually, the sine of the angle is a better measure since it will range from 0 when the vectors are closest (most nearly parallel) to 1 when they are perpendicular without our having to worry about the actual angles or their signs.

The Number of Features in Common

When the preponderance of fields in the records we wish to compare are categorical variables, we abandon geometric measures and turn instead to measures based on the degree of overlap between records. As with the

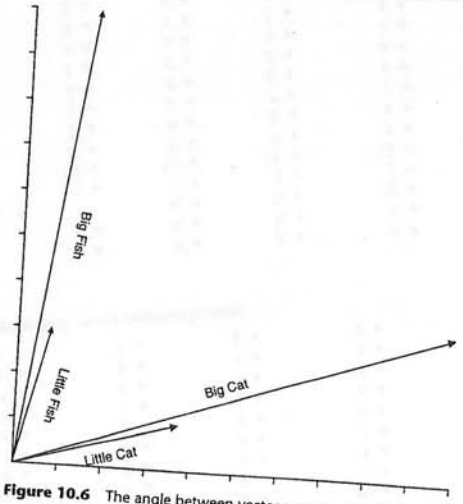

Figure 10.6 The angle between vectors as a measure of association.

geometric measures, there are many variations on this idea. In all variations, we compare two records field by field and count the number of fields that match and the number of fields that don't match. The simplest measure is the ratio of matches to the total number of fields.

In its simplest form, this measure counts two null fields as matching with the result that everything we don't know much about ends up in the same cluster. A simple improvement is to not include matches of this sort in the match count. Other variations are useful under various conditions. If the usual degree of overlap is high, you can give extra weight to unmatched fields by counting them double so that any minor mismatch is punished. If, on the other hand, the usual degree of overlap is low, you can give extra weight to matches to make sure that even a small overlap is rewarded.

The most sophisticated measures weight the matches by the prevalence of each class in the general population. After all, a match on "1956 Chevy Nomad" ought to count for more than a match on "1997 Ford Taurus."

What K Means

Clusters form some subset of the field variables tend to vary together. If all the variables are truly independent, no clusters will form—the entire space will be filled with an even haze of data points. At the opposite extreme, if all the variables are dependent on the same thing (in other words, if they are co-linear), then all the records will form a single cluster. In between these extremes, we don't really know how many clusters to expect. If we go looking for a certain number of clusters, we may find them. But that doesn't mean that there aren't other perfectly good clusters lurking in the data where we could find them by trying a different value of K.

In his excellent 1973 book, *Cluster Analysis for Applications*, Michael Anderberg uses a deck of playing cards to illustrate many aspects of clustering. We have borrowed his idea to illustrate the way that the initial choice of K, the number of cluster seeds, can have a large effect on the kinds of clusters that will be found.

In descriptions of K-means and related algorithms, the selection of K is often glossed over. But since, in many cases, there is no a priori reason to select a particular value, there is really an outermost loop to these algorithms that occurs in the analyst rather than in the computer program. This outer loop consists of performing automatic cluster detection using one value of K, evaluating the results, then trying again with another value of K.

After each trial, the strength of the resulting clusters can be evaluated by comparing the average distance between records in a cluster with the average distance between clusters, and by other procedures described later in this chapter. But the clusters must also be evaluated on a more subjective basis to determine their usefulness for a given application. As shown in Figures 10.7, 10.8, 10.9, 10.10, and 10.11, it is easy to create very good clusters from a deck of playing cards using various values for K and various distance measures. In the case of

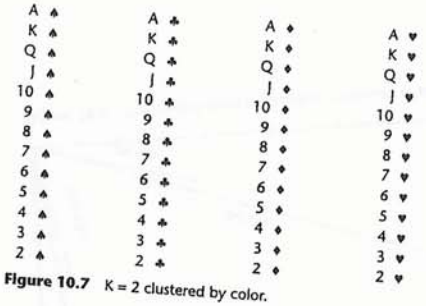

Figure 10.7 K = 2 clustered by color.

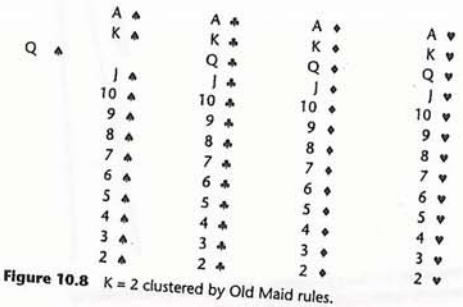

Figure 10.8 K = 2 clustered by Old Maid rules.

A ♠	A ♣	A ♦	A ♥
K ♠	K ♣	K ♦	K ♥
Q ♠	Q ♣	Q ♦	Q ♥
J ♠	J ♣	J ♦	J ♥
10 ♠	10 ♣	10 ♦	10 ♥
9 ♠	9 ♣	9 ♦	9 ♥
8 ♠	8 ♣	8 ♦	8 ♥
7 ♠	7 ♣	7 ♦	7 ♥
6 ♠	6 ♣	6 ♦	6 ♥
5 ♠	5 ♣	5 ♦	5 ♥
4 ♠	4 ♣	4 ♦	4 ♥
3 ♠	3 ♣	3 ♦	3 ♥
2 ♠	2 ♣	2 ♦	2 ♥

Figure 10.9 K = 2 clustered by rules for War, Beggar My Neighbor, and many other games.

	A ♠	A ♣	A ♦	A ♥
	K ♠	K ♣	K ♦	K ♥
Q ♠		Q ♣	Q ♦	Q ♥
	J ♠	J ♣	J ♦	J ♥
	10 ♠	10 ♣	10 ♦	10 ♥
	9 ♠	9 ♣	9 ♦	9 ♥
	8 ♠	8 ♣	8 ♦	8 ♥
	7 ♠	7 ♣	7 ♦	7 ♥
	6 ♠	6 ♣	6 ♦	6 ♥
	5 ♠	5 ♣	5 ♦	5 ♥
	4 ♠	4 ♣	4 ♦	4 ♥
	3 ♠	3 ♣	3 ♦	3 ♥
	2 ♠	2 ♣	2 ♦	2 ♥

Figure 10.10 K = 3 clustered by rules for Hearts.

A ♠	A ♣	A ♦	A ♥
K ♠	K ♣	K ♦	K ♥
Q ♠	Q ♣	Q ♦	Q ♥
J ♠	J ♣	J ♦	J ♥
10 ♠	10 ♣	10 ♦	10 ♥
9 ♠	9 ♣	9 ♦	9 ♥
8 ♠	8 ♣	8 ♦	8 ♥
7 ♠	7 ♣	7 ♦	7 ♥
6 ♠	6 ♣	6 ♦	6 ♥
5 ♠	5 ♣	5 ♦	5 ♥
4 ♠	4 ♣	4 ♦	4 ♥
3 ♠	3 ♣	3 ♦	3 ♥
2 ♠	2 ♣	2 ♦	2 ♥

Figure 10.11 K = 4 clustered by suit.

playing cards, the distance measures are dictated by the rules of various games. The distance from Ace to King, for example, might be 1 or 12 depending on the game.

K = N?

Even with playing cards, some values of K don't lead to good clusters—at least not with distance measures suggested by the card games known to the authors. There are obvious clustering rules for K = 1, 2, 3, 4, 8, 13, 26, and 52. For these values we can come up with "perfect" clusters where each element of a cluster is equidistant from every other member of the cluster, and equally far away from the members of some other cluster. For other values of K, we have the more familiar situation that some cards do not seem to fit particularly well in any cluster.

The Importance of Weights

It is important to differentiate between the notions of *scaling* and *weighting*. They are not the same, but they are often confused. Scaling deals with the problem that different variables are measured in different units. Weighting deals with the problem that we care about some variables more than others.

In geometry, all dimensions are equally important. Two points that differ by 2 in dimensions X and Y and by 1 in dimension Z are the

same distance from one another as two other points that differ by 1 in dimension X and by 2 in dimensions Y and Z. We don't even ask what units X, Y, and Z are measured in; it doesn't matter, so long as they are the same.

But what if X is measured in yards, Y is measured in centimeters, and Z is measured in nautical miles? A difference of 1 in Z is now equivalent to a difference of 185,200 in Y or 2,025 in X. Clearly, they must all be converted to a common scale before distances will make any sense.

Unfortunately, in commercial data mining there is usually no common scale available because the different units being used are measuring quite different things. If we are looking at plot size, household size, car ownership, and family income, we cannot convert all of them to acres or dollars. On the other hand, it seems bothersome that a difference of 20 acres in plot size is indistinguishable from a change of $20 in income. The solution is to map all the variables to a common *range* (often 0 to 1 or –1 to 1). That way, at least the ratios of change become comparable—doubling plot size will have the same effect as doubling income. We refer to this remapping to a common range as *scaling*.

But what if we think that two families with the same income have more in common than two families on the same size plot, and we want that to be taken into consideration during clustering? That is where weighting comes in.

TIP

Here are three common ways of scaling variables to bring them all into comparable ranges:

1. Divide each variable by the mean of all the values it takes on.
2. Divide each variable by the range (the difference between the lowest and highest value it takes on) after subtracting the lowest value.
3. Subtract the mean value from each variable and then divide by the standard deviation. This is often called "converting to z scores."

Use Weights to Encode Outside Information

Scaling takes care of the problem that changes in one variable appear more significant than changes in another simply because of differences in the speed with which the units they are measured in get incremented. Many books recommend scaling all variables to a normal form with a mean of zero and a variance of one. That way, all fields contribute equally when the distance between two records is computed.

We suggest going farther. The whole point of automatic cluster detection is to find clusters that make sense to *you*. If, for your purposes, whether people have children is much more important than the number of credit cards they carry, there is no reason not to bias the outcome of the clustering by multiplying the number of children field by a higher weight than the number of credit cards field. After scaling to get rid of bias that is due to the units, you should use weights to introduce bias based on your knowledge of the business context.

Of course, if you want to evaluate the effects of different weighting strategies, you will have to add yet another outer loop to the clustering process. In fact, choosing weights is one of the optimization problems that can be addressed with genetic algorithms as discussed in Chapter 14.

Variations on the K-Means Method

The basic K-means algorithm has many variations. It is likely that the commercial software tools you find to do automatic clustering will incorporate some of these variations. Among the differences you are likely to encounter are:

- Alternate methods of choosing the initial seeds
- Alternate methods of computing the next centroid
- Using probability density rather than distance to associate records with clusters.

Of these, only the last is important enough to merit further discussion here.

Gaussian Mixture Models

The K-means method as we have described it has some drawbacks.

- It does not do well with overlapping clusters.
- The clusters are easily pulled off center by outliers.
- Each record is either in or out of a cluster; there is no notion of some records being more or less likely than others to really belong to the cluster to which they have been assigned.

Gaussian mixture models are a probabilistic variant of K-means. Their name comes from the Gaussian distribution, a probability distribution often assumed for high-dimensional problems. As before, we start by choosing K seeds. This time, however, we regard the seeds as means of Gaussian distributions. We then iterate over two steps called the estimation step and the maximization step.

In the estimation step, we calculate the *responsibility* that each Gaussian has for each data point (see Figure 10.12). Each Gaussian has strong responsibility for points that are close to it and weak responsibility for points that are distant. The responsibilities will be used as weights in the next step.

In the maximization step, the mean of each Gaussian is moved towards the centroid of the entire data set, *weighted by the responsibilities* as illustrated in Figure 10.13.

These steps are repeated until the Gaussians are no longer moving. The Gaussians themselves can grow as well as move, but since the

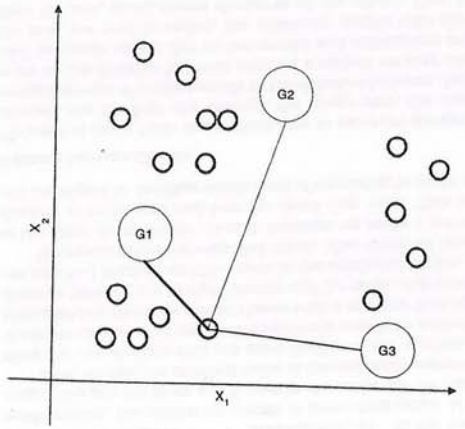

Figure 10.12 In the estimation step, each Gaussian is assigned some responsibility for each point. Thicker lines indicate greater responsibility.

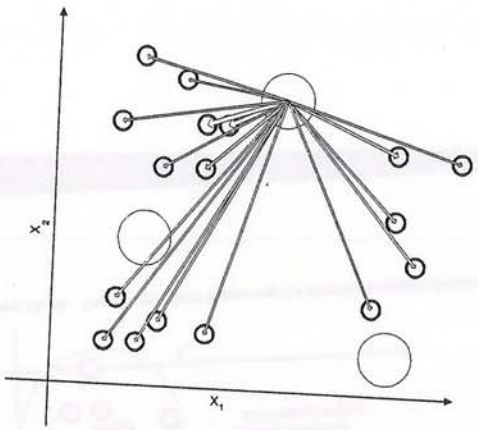

Figure 10.13 Each Gaussian mean is moved to the centroid of all the data points weighted by the responsibilities for each point. Thicker arrows indicate higher weights.

distribution must always integrate to one, a Gaussian gets weaker as it gets bigger. Responsibility is calculated in such a way that a given point may get equal responsibility from a nearby Gaussian with low variance and from a more distant one with higher variance.

The reason this is called a "mixture model" is that the probability at each data point is the sum of a mixture of several distributions. At the end of the process, each point is tied to the various clusters with higher or lower probability. This is sometimes called *soft clustering*.

AGGLOMERATION METHODS

In the K-means approach to clustering, we start out with a fixed number of clusters and gather all records into them. There is another class of methods that work by agglomeration. In these methods, we start out with each data point forming its own cluster and gradually merge clus-

ters until all points have been gathered together in one big cluster. Towards the beginning of the process, the clusters are very small and very pure—the members of each cluster are few, but very closely related. Towards the end of the process, the clusters are large and less well-defined. The entire history is preserved so you can choose the level of clustering that works best for your application.

The Agglomerative Algorithm

The first step is to create a *similarity matrix*. The similarity matrix is a table of all the pair-wise distances or degrees of association between points. As before, we can use any of a large number of measures of association between records, including the Euclidean distance, the angle between vectors, and the ratio of matching to nonmatching categorical fields. The issues raised by the choice of distance measures are exactly the same as previously discussed in relation to the K-means approach.

At first glance you might think that if we have N data points we will need to make N^2 measurements to create the distance table, but if we assume that our association measure is a true distance metric, we actually only need half that because all true distance metrics follow the rule that `Distance(X,Y) = Distance(Y,X)`. In the vocabulary of mathematics, the similarity matrix is lower triangular. At the beginning of the process there are N rows in the table, one for each record.

Next, we find the smallest value in the similarity matrix. This identifies the two clusters that are most similar to one another. We merge these two clusters and update the similarity matrix by replacing the two rows that described the parent cluster with a new row that describes the distance between the merged cluster and the remaining clusters. There are now N–1 clusters and N–1 rows in the similarity matrix.

We repeat the merge step N–1 times, after which all records belong to the same large cluster. At each iteration we make a record of which clusters were merged and how far apart they were. This information will be helpful in deciding which level of clustering to make use of.

Distance between Clusters

We need to say a little more about how to measure distance between clusters. On the first trip through the merge step, the clusters to be merged consist of single records so the distance between clusters is the same as the distance between records, a subject we may already have said too much about. But on the second and subsequent trips around the loop, we need to update the similarity matrix with the distances from the new, multi-record cluster to all the others. How do we measure this distance?

As usual, there is a choice of approaches. Three common ones are:

- Single linkage
- Complete linkage
- Comparison of centroids

In the single linkage method, the distance between two clusters is given by the distance between the *closest* members. This method produces clusters with the property that every member of a cluster is more closely related to at least one member of its cluster than to any point outside it.

In the complete linkage method, the distance between two clusters is given by the distance between their *most distant* members. This method produces clusters with the property that all members lie within some known maximum distance of one another.

In the third method, the distance between two clusters is measured between the centroids of each. The centroid of a cluster is its average element. Figure 10.14 gives a pictorial representation of all three methods.

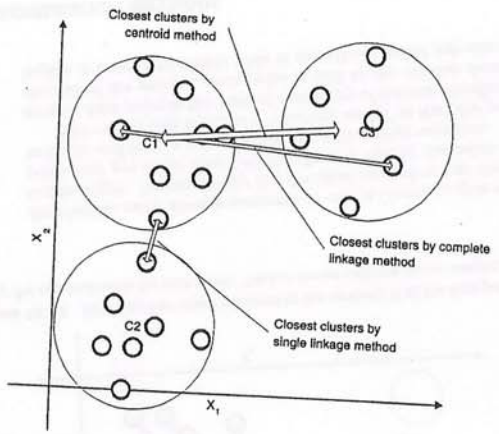

Figure 10.14 Three methods of measuring the distance between clusters.

Clusters and Trees

The agglomeration algorithm creates hierarchical clusters. At each level in the hierarchy, clusters are formed from the union of two clusters at the next level down. Another way of looking at this is as a tree, much like the decision trees discussed in Chapter 12 except that cluster trees are built by starting from the leaves and working towards the root.

Clustering People by Age: An Example of Agglomerative Clustering

To illustrate agglomerative clustering, we have chosen an example of clustering in one dimension using the single linkage measure for distance between clusters. These choices should enable you to follow the algorithm through all its iterations in your head without having to worry about squares and square roots.

The data consists of the ages of people at a family gathering. Our goal is to cluster the participants by age. Our metric for the distance between two people is simply the difference in their ages. Our metric for the distance between two clusters of people is the difference in age between the oldest member of the younger cluster and the youngest member of the older cluster. (The one dimensional version of the single linkage measure.)

Because the distances are so easy to calculate, we dispense with the similarity matrix. Our procedure is to sort the participants by age, then begin clustering by first merging clusters that are 1 year apart, then 2 years, and so on until there is only one big cluster.

Figure 10.15 shows the state of the clusters after six iterations, with three clusters remaining. This is the level of clustering that seems the most useful. The algorithm appears to have clustered the population into three generations.

EVALUATING CLUSTERS

When using the K-means approach to cluster detection, we need a way to determine what value of K finds the best clusters. Similarly, when using a hierarchical approach, we need a test for which level in the hierarchy contains the best clusters. But what does it mean to say that a cluster is good?

In general terms, we want clusters whose members have a high degree of similarity—or in geometric terms, are close to each other—and we want the clusters themselves to be widely spaced.

A standard measure of the within-cluster similarity is the variance—the sum of the squared differences of each element from the mean, so we might simply look for the solutions that produce the clusters with the lowest variance. But for hierarchical clustering, this does not make sense since we always start out with clusters of one which have no variance at all. A good measure to use with hierarchical clusters is the difference between the distance value at which it was formed and the distance value at which it is merged into the next level. Strong clusters, like the one linking 1 to 13-year-olds at distance 3 in Figure 10.15, last a long time.

A general-purpose measure that works with any form of cluster detection is to take whatever similarity measure or distance metric you used to form the clusters and use it to compare the average distance within clusters to the average distance between clusters. This can be done for each cluster individually and for the entire collection of clusters.

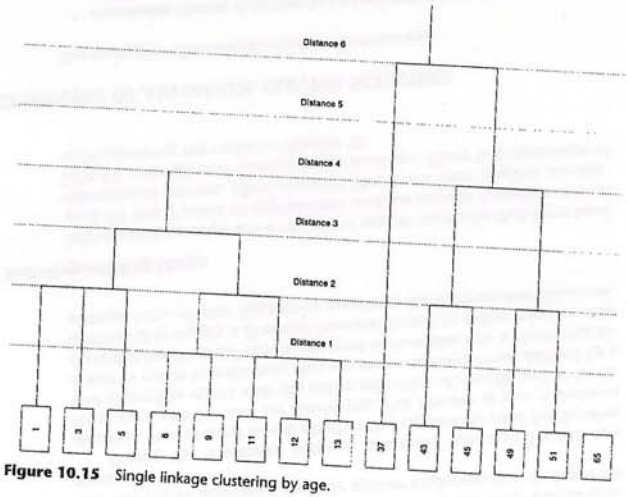

Figure 10.15 Single linkage clustering by age.

TIP

If you find that there are one or two good clusters along with a number of weaker ones, you may be able to improve your results by removing all members of the strong clusters. You will want to isolate the strong clusters for further analysis anyway, and removing their strong pull may allow new clusters to be detected in the records left behind.

Inside the Cluster

Once you have found a strong cluster, you will want to analyze what makes it special. What is it about the records in this cluster that causes them to be lumped together? Even more importantly, is it possible to find rules and patterns within this cluster now that the noise from the rest of the database has been eliminated?

The easiest way to approach the first question is to take the mean of each variable within the cluster and compare it to the mean of the same variable in the parent population. Rank order the variables by the magnitude of the difference. Looking at the variables that show the largest difference between the cluster and the rest of the database will go a long way towards explaining what makes the cluster special.

As for the second question, that is what all the other data mining techniques are for!

Outside the Cluster

Clustering can be useful even when only a single cluster is found. When screening for a very rare defect, there may not be enough examples to train a directed data mining model to detect it. One example is testing electric motors at the factory where they are made. Cluster detection methods can be used on a sample containing only good motors to determine the shape and size of the "normal" cluster. When a motor comes along that falls outside the cluster for any reason, it is suspect. This approach has been used in medicine to detect the presence of abnormal cells in tissue samples.

OTHER APPROACHES TO CLUSTER DETECTION

In addition to the two approaches to automatic cluster detection described in this chapter, there are two other approaches that make use

of variations of techniques discussed in Chapters 12 and 13—decision trees and neural networks.

Divisive Methods

We have already noted the similarity between the tree formed by the agglomerative clustering techniques and the ones formed by decision tree algorithms such as CART and CHAID. Although the agglomerative methods work from the leaves to the root, while the decision tree algorithms work from the root to the leaves, they both create a similar hierarchical structure. The hierarchical structure reflects another similarity between the methods. Decisions made early on in the process are never revisited, which means that some fairly simple clusters will not be detected if an early split or agglomeration destroys the structure.

Seeing the similarity between the trees produced by the two methods, it is natural to ask whether the algorithms used for decision trees may also be used for clustering. The answer is yes. A decision tree algorithm starts with the entire collection of records and looks for a way to spit it into clusters that are *purer*, in some sense defined by a diversity function. All that is required to turn this into a clustering algorithm is to supply a diversity function chosen to either minimize the average intra-cluster distance or maximize the inter-cluster distances.

Self-Organizing Maps

Self-organizing maps are a variant of neural networks that have been used for many years in applications such as feature detection in two-dimensional images. More recently, they have been applied successfully for more general clustering applications. There is a discussion of self-organizing networks in Chapter 13.

STRENGTHS OF AUTOMATIC CLUSTER DETECTION

The strengths of automatic cluster detection are:

- Automatic cluster detection is an undirected knowledge discovery technique.
- Automatic cluster detection works well with categorical, numeric, and textual data.
- Easy to apply.

Automatic Cluster Detection Is Undirected

The chief strength of automatic cluster detection is that it is undirected. This means that it can be applied even when you have no prior knowledge of the internal structure of a database. Automatic cluster detection can be used to uncover hidden structure that can be used to improve the performance of more directed techniques.

Clustering Can Be Performed on Diverse Data Types

By choosing different distance measures, automatic clustering can be applied to almost any kind of data. It is as easy to find clusters in collections of news stories or insurance claims as in astronomical or financial data.

Automatic Cluster Detection Is Easy to Apply

Most cluster detection techniques require very little massaging of the input data and there is no need to identify particular fields as inputs and others as outputs.

WEAKNESSES OF AUTOMATIC CLUSTER DETECTION

Weaknesses of this technique are:

- It can be difficult to choose the right distance measures and weights.
- Sensitivity to initial parameters.
- It can be hard to interpret the resulting clusters.

Difficulty with Weights and Measures

The performance automatic cluster detection algorithms is highly dependent on the choice of a distance metric or other similarity measure. It is sometimes quite difficult to devise distance metrics for data that contains a mixture of variable types. It can also be difficult to determine a proper weighting scheme for disparate variable types.

Sensitivity to Initial Parameters

In the K-means method, the original choice of a value for K determines the number of clusters that will be found. If this number does not match the natural structure of the data, the technique will not obtain good results.

Difficulty Interpreting Results

A strength of automatic cluster detection is that it is an undirected knowledge discovery technique. The flip side is that when you don't know what you are looking for, you may not recognize it when you find it! The clusters you discover are not guaranteed to have any practical value.

WHEN TO USE CLUSTERING

Clustering is a great tool to use when you are faced with a large, complex data set with many variables and a lot of internal structure. At the start of a new data mining project, clustering is often the best first technique to turn to. It is rarely the only tool, however. Once automatic cluster detection has discovered regions of the data space that contain similar records, other data mining tools have a better chance of discovering rules and patterns within them.