3

Distance Function Learning for Supervised Similarity Assessment

A. Bagherjeiran and C.F. Eick

Department of Computer Science
University of Houston
Houston, TX 77204-3010, USA
{abagherj,ceick}@cs.uh.edu

Summary. Assessing the similarity between cases is a prerequisite for many case-based reasoning tasks. This chapter centers on distance function learning for supervised similarity assessment. First a framework for supervised similarity assessment is introduced. Second, three supervised distance function learning approaches from the areas of pattern classification, supervised clustering, and information retrieval are discussed, and their results for two supervised learning tasks will be explained and visualized. In each of these different areas, we show how the method can be applied to areas of case-based reasoning. Finally, a detailed literature survey will be given.

3.1 Introduction

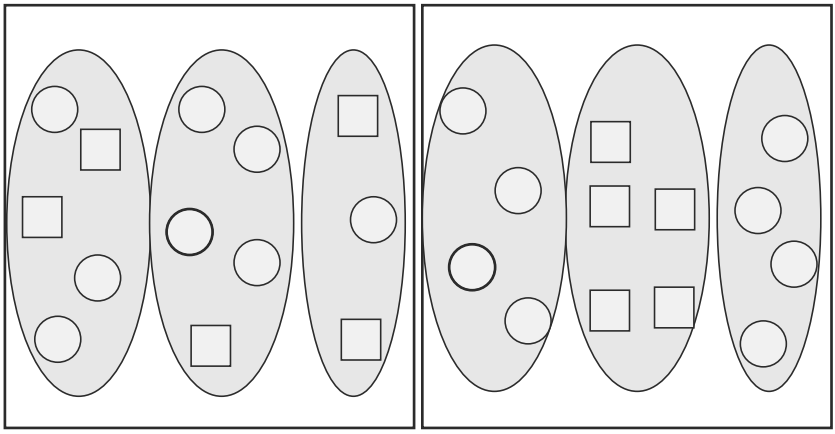
Case-based reasoning depends heavily on assessing the similarity between cases. Similarity assessment is the task of determining which cases are similar to each other and which are dissimilar. In general, defining distance functions is a difficult and tedious task.

There are several uses of distance functions in case-based reasoning all of which can benefit from distance function learning. In case-base classification, cases belonging to the same class should have a lower distance than cases in different classes. A good distance function is one that leads to a highly accurate classifier. In case base maintenance, the case base is reorganized into clusters so that search time is reduced by consulting the cluster's representative first. Cases far from the representative of the cluster can be removed to improve search time. A good distance function is one that leads to tight and cohesive clusters, effectively organizing the case base. In case-based retrieval, several cases relevant to a user's query are returned and users send feedback to improve retrieval. A good distance function is one that retrieves a variety of relevant cases for a user's query. Each of these applications places different requirements on the distance function, which makes manually tuning the distance function difficult.

A. Bagherjeiran and C.F. Eick: *Distance Function Learning for Supervised Similarity Assessment*, Studies in Computational Intelligence (SCI) **73**, 91–126 (2008)
www.springerlink.com

Due to the fact that distance functions are difficult to design for different applications, it is worthwhile to look for approaches that learn distance functions from the case base. In particular this chapter will discuss methods in the context of classification, where the goal of supervised similarity assessment is to obtain distance functions that separate cases belonging to different classes well. Consequently, the scope of the methods discussed in this chapter is limited to case bases that have class labels. Class labels either represent natural classes or capture information about the relevancy of a particular case for a particular use. Different methods are needed for unsupervised learning tasks, such as clustering. The class labels associated with different cases can be viewed as feedback that, as we will see, is instrumental for tailoring distance functions for a particular classification task.

Figure 3.1 illustrates what distance function learning for supervised similarity assessment is trying to accomplish; it depicts the distances of 13 cases, five of which belong to a class that is identified by a square and eight belong to a different class that is identified by a circle. When using the initial distance function d_{init} we do not observe much clustering with respect to the two classes. Starting from this distance function, we would like to obtain a better distance function d_{good} so that the cases belonging to the same class are clustered together. In Fig. 3.1 we can identify three clusters with respect to d_{good}, two containing only circles and one containing only squares. Why is it beneficial to find such a distance function d_{good}? Most importantly, using the learned distance function in conjunction with a k-nearest neighbor classifier [10] allows us to obtain a classifier with high predictive accuracy. For example, if we use a three-nearest neighbor classifier with d_{good} it will have 100% accuracy with respect to leave-one-out cross-validation, whereas several cases are misclassified if d_{init} is used. The second advantage is that looking



a) d_{init} b) d_{good}

Fig. 3.1. Objective of supervised distance function learning

at d_{good} itself will tell us which features are important for the particular classification problem.

It is also important to understand that there are no universal distance functions for a given case base. The usefulness of a distance function is determined in the context of the task it is used for. Consequently, a distance function that does a good job in separating cancer patients from healthy patients is unlikely to be useful in separating diabetes patients from healthy patients.

This chapter surveys how distance function learning is performed in three separate areas of case-based reasoning: classification, maintenance, and retrieval. Section 3.2 reviews basic properties of distance functions and introduces a supervised distance function learning framework. Section 3.3 discusses the Relief algorithm used in case-based classification [30]. Section 3.4 discusses the inside–outside weight adjustment algorithm used as a supervised method for case-base maintenance [17]. Section 3.5 discusses a relevance feedback algorithm used in case-based retrieval [37]. In Sect. 3.6 we show how, on two example case bases, to visualize a distance function before and after learning. Section 3.7 surveys recent work in distance function learning. Section 3.8 gives a brief summary.

3.2 Distance Function Learning Model

A case x consists of a vector of features and a class label (target value). We refer to x as one case in a space of possible cases X, as $x \in X$ and $class(x)$ as the class to which x belongs.

3.2.1 Distance Functions

In general, the case need not be a vector. We define a distance function over a set of cases $x \in X$. A distance function $d : X \times X \to \mathbb{R}$ should satisfy the following two conditions for $x, y, z \in X$:

1. If $d(x, y) = 0$, then x and y are the same or as similar as possible.
2. If x is closer to y than it is to z, then $d(x, y) < d(x, z)$.

A few more restrictions are required for practical distance functions.

Definition 1. *A function $d : X \times X \to \mathbb{R}$ is a* **metric** *on the space X if for all $x, y, z \in X$ the following conditions hold*:

$$0 \leq d(x, y) < \infty$$

$$d(x, y) = d(y, x) \tag{3.1}$$

$$d(x, y) = 0 \iff x = y \tag{3.2}$$

$$d(x, z) \leq d(x, y) + d(x, z) \tag{3.3}$$

The pair (X, d) *is called a* **metric space**.

Condition (3.2) implies that two cases with distance 0 are equivalent. Duplicate cases occur in practice and, either because of rounding or preprocessing, they may have a distance of 0 even if the cases are not the same. A simple application of the Condition (3.3) is the saying that the shortest distance between two points is a straight line. This is because the Euclidean distance is a metric in $\mathbb{R}^2$.

Norms

A metric depends only on the definition of the distance function and not on the composition of the set. In practice, however, the set X is the vector space $\mathbb{R}^n$ where each $x \in X$ is a vector of length n:

$$x = (x_1, \ldots, x_n)$$

where $x_j \in \mathbb{R}$ for all $1 \leq j \leq n$. Each element, x_j, in the vector is referred to as a feature, and the vector x is called the feature vector. Given this vector definition of a case, we say that a distance function is imposed by a norm defined over the vector space.

Definition 2. *A function* $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}_0^+$ *is a* **norm** *if for all* $x, y \in \mathbb{R}^n$ *and* $\alpha \in \mathbb{R}$*, the following conditions hold*:

$$\begin{aligned} \|x + y\| &\leq \|x\| + \|y\| \\ \|\alpha x\| &= |\alpha| \, \|x\| \\ \|x\| &> 0 \text{ for } x \neq 0 \end{aligned}$$

The first condition is known as the *triangle inequality*. Given a norm $\|\cdot\|$ over $\mathbb{R}^n$, the following distance function is a metric:

$$d(x, y) = \|x - y\|$$

where $x, y \in \mathbb{R}^n$.

We define a family of norms and distance functions.

Definition 3. *A function* $\|\cdot\|_p : \mathbb{R}^n \to \mathbb{R}_0^+$ *is the* L_p **norm** *if for all* $p \in \mathbb{R}$*,* $p \geq 1$*, and* $x \in \mathbb{R}^n$

$$\|x\|_p = \left(\sum_{j=1}^{n} |x_j|^p \right)^{1/p}$$

For each L_p norm, there is a corresponding L_p distance function metric.

Definition 4. *The function* $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^+$ *is the* L_p **distance metric** *if for all* $p \in \mathbb{R}$, $p \geq 1$, *and* $x, y \in \mathbb{R}^n$

$$d(x, y) = \|x - y\|_p$$

where $\|\cdot\|_p$ *is the* L_p *norm.*

This family of distance functions is also known as the Minkowski distance. There are three common p values, which correspond to the following three distance functions:

L_1 Manhattan or city-block distance:

$$d(x, y) = \sum_{j=1}^{n} |x_j - y_j|$$

L_2 Euclidean distance:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2}$$

L_∞ Max distance:

$$d(x, y) = \max_{j=1}^{n} |x_j - y_j|$$

In the remainder of this chapter we consider only the L_1 distance because the notation is simplest and it is faster to compute than L_2.

3.2.2 Parameterized Distance Functions

In distance function learning, the objective is to find parameter values for a parameterized distance function. The most common approach to obtain a parameterized distance function is feature weighting. Feature weights are the output of the distance function learning algorithms discussed here. Weighted distance functions are a subclass of the more general linear distance functions.

Feature weights, one for each attribute, emphasize one subset of features over the others. For the L_1 distance, the weighted version is

$$d_w(x, y) = \sum_{j=1}^{n} w_j |x_j - y_j| \tag{3.4}$$

where $x, y, w \in \mathbb{R}^n$. We place a few restrictions on the vector w_j for $1 \leq j \leq n$:

$$\begin{aligned} w &= (w_1, \ldots, w_n) \\ w_j &\geq 0 \\ \sum_{j=1}^{n} w_j &= 1 \end{aligned}$$

These restrictions simplify the programming. In addition, positive weights maintain the semantics of a distance function since negative weights and distances have no clear meaning.

Linear distance functions generalize the L_2 distance by projecting the original distance vector onto a linear basis A:

$$d_A(x, y) = (x - y)^T A (x - y)$$

where x, y are column vectors and A is an $n \times n$ projection matrix. Although most definitions of linear distance function take the square root of $d_A(x, y)$, we usually ignore the root because it is expensive to compute and does not change the relative distances. When A is positive semi-definite, the distance $d_A(x, y)$ is a distance metric. When A is the identity matrix ($I_{n\times n}$), the distance reduces to the squared L_2 distance as shown:

$$\begin{aligned} d_I(x, y) &= (x - y)^T I_{n\times n} (x - y) \\ &= (x - y)^T (x - y) \\ &= \sum_{j=1}^{n} (x_j - y_j)^2 \\ &= \|x - y\|^2 \end{aligned}$$

For small- to medium-sized case bases, it is often more efficient to compute and store the distances between all pairs of cases. This forms a distance matrix for the case base X and distance function d.

Definition 5. *A matrix $D \in \mathbb{R}^{m\times m}$ is a* **distance matrix** *if and only if:*

$$D_{i,j} = d(x_i, x_j)$$

for $1 \leq i, j \leq m$, where d is a distance function, $x_i, x_j \in X$, and m is the number of cases in X.

For distance function learning, we often decompose the distance matrix into the set of matrices D_j for $1 \leq j \leq n$, storing the distance with respect to the jth feature. A weighted distance matrix for the L_1 distance can be recreated as $D = \sum_{j=1}^{n} w_j D_j$ for feature weights w. For a metric distance function, D is a symmetric matrix with zeros along the diagonal.

Examples

Example 1. A distance function without weights is assumed to have weights, but they are equal and nonzero:

$$w_j = \frac{1}{n}$$

for $1 \leq j \leq n$.

Example 2. As a linear distance function, a weighted distance function consists of a diagonal projection matrix $W = \text{diag}(w)$:

$$\begin{aligned} d_W(x, y) &= (x - y)^T W (x - y) \\ &= \sum_{j=1}^{n} w_j (x - y)^2 \end{aligned} \tag{3.5}$$

where w is a weight vector as we have previously described. Although more complex, we can obtain a linear form for the L_1 distance:

$$\begin{aligned} d_W(x, y) &= \sqrt{|x - y|}^T W \sqrt{|x - y|} \\ &= \sum_{j=1}^{n} w_j |x_j - y_j| \end{aligned} \tag{3.6}$$

where $\sqrt{\cdot}$ denotes the component-wise root. In practice, it is usually more efficient to use the form in (3.4) or (3.5).

Example 3. A common linear distance function is the Mahalanobis distance.

$$d_\Sigma(x, y) = (x - y)^T \Sigma^{-1} (x - y)$$

where Σ is the $n \times n$ covariance matrix. This is often referred to the Mahalanobis distance between x and y. It assumes that x and y are distributed according to the multivariate normal distribution $N(\mu, \Sigma)$ with mean μ. Since both vectors are in the same distribution, the effect of the mean cancels are as shown:

$$\begin{aligned} d_\Sigma(x, y) &= ((x - \mu) - (y - \mu))^T \Sigma^{-1} ((x - \mu) - (y - \mu)) \\ &= ((x - y - \mu + \mu))^T \Sigma^{-1} ((x - y - \mu + \mu)) \\ &= (x - y)^T \Sigma^{-1} (x - y) \end{aligned}$$

3.3 Case-Based Classification

In case-based classification, each case has a class label. The objective of a classification algorithm is to determine the class for an incoming case. In classification, an ideal distance function should make cases close to other cases in the same class, while making them far from cases belonging to a different

class. Most distance function learning algorithms for classification find parameters for a parameterized distance function and then apply an existing distance-based classifier.

A classifier that uses distance functions is the k-nearest neighbor (k-NN) algorithm [10]. In pattern classification, we consider a case base $Y \subseteq X$ on which learning is performed. The subset Y is further divided into a training set $Tr \subset Y$ and a testing set $Te \subset Y$ such that $Tr \cup Te = Y$ and $Tr \cap Te = \emptyset$. Given a distance function d and a training set of cases Tr, the 1-NN algorithm assigns the class label to a new case in the testing set $y \in Te$ as follows based on its nearest neighbor in the training set:

$$class(y) = class(x^\star)$$
$$x^\star = \arg\min_{x \in Tr} d_w(x, y)$$

The case $x^\star \in Tr$ is called the nearest neighbor to y with respect to the distance function d. Despite its simplicity, the k-NN algorithm is usually accurate and fast for medium-sized training sets.

3.3.1 Feature Evaluation

Two feature evaluation methods are common in the literature. Feature selection algorithms finds a subset of features that more compactly represent the case base. Feature evaluation algorithms assigns a score to each feature that indicates the degree to which it is useful for classification. The higher the score, the more useful a feature is. Thus, feature selection algorithms are a specialization of feature evaluation methods, in which the score is either 0 or 1. In high-dimensional case bases, such as images or signals, feature evaluation methods are used to remove features that have a low score. This sometimes improves classification accuracy and reduces computation time.

We use feature evaluation algorithms to compute weights in a distance function. We make the weights proportional to the score. Let $s = (s_1, \ldots, s_n)$ be the scores for each of the n features. We compute the weight vector w as

$$w_i = \frac{s_j}{\sum_{j=1}^{n} s_j}$$

where $s_j \geq 0$ for $1 \leq j \leq n$.

3.3.2 Relief-F

The Relief algorithm was originally designed as a feature evaluation algorithm, but it is commonly used to compute weights for distance functions [28]. The weight of a feature corresponds to how well it separates cases from different classes while not separating cases from the same class. Weights are updated for each case in the case base.

Algorithm 1 Relief-F's weight-update where $class(X)$ is the set of all classes in the space X and $p(C)$ is the proportion of cases in the training set that belong to class C. This update yields a new weight vector, and is repeated until the weights converge.

Input: Training set Tr , weight vector w, random sample size m.
Output: Updated weight vector w'

1. Select m cases $x \in T$ from Tr randomly with uniform probability.
2. $w' = w$
3. For $1 \leq i \leq m$
   a) For $C \in class(X)$
      M_i^C = Closest case to x_i in class C using old weights w
   b) $H_i = M_i^{class(x_i)}$
   c) For $1 \leq j \leq n$
      Update weight for feature j

$$w'_j = w'_j - \frac{1}{m}(x_{i,j} - H_{i,j}) + \frac{1}{m} \sum_{C \neq class(x_i)} p(C)(x_{i,j} - M^C_{i,j}) \quad (3.7)$$

Algorithm 1 shows the steps in a single iteration of the Relief-F algorithm, which is an extension of the original Relief algorithm designed to handle multiple classes [30]. From the training set $Tr \subset Y$, a random sample ($T \subseteq Tr$) of size m is selected. The weight of each feature, w'_j, is updated for each case x_i in T. The update considers the nearest neighbor of x_i from each class using the last weight vector, w. The neighbor in the same class as x_i, H_i, is called the hit case. Conversely, the nearest neighbor in each class $C \neq class(x_i)$, M_i^C, is called a miss case. The update decreases the feature weight to bring x_i closer to the feature value of the hit and increases the weight to move x_i away from the feature value of the misses. This weight update procedure is repeated for the new weight vector w' until the weights converge.

Relief does not have an explicit objective function for the weights. Weights are updated based on their prediction error, which makes this an iterative hill-climbing approach. By updating weights for individual cases, Relief utilizes local information. Many other feature evaluation methods consider only global information such as the information gain (entropy with respect to class) and the χ^2 statistic (variance in the feature values) [16]. Local information is particularly useful in distance function learning.

Example 4. Consider a set Tr of four cases in two classes completely separable along the first dimension, which we will call the x-axis. For convenience, the distance matrix is decomposed:

$$D_w = w_1 D_1 + w_2 D_2$$

where D_i is the distance matrix with respect to feature i and w is the weight vector. Let D_1 and D_2 be the feature-specific distance matrices for the four cases:

$$D_1 \qquad D_2$$
$$\begin{bmatrix} 0\,0\,1\,1 \\ 0\,0\,1\,1 \\ 1\,1\,0\,0 \\ 1\,1\,0\,0 \end{bmatrix} \quad \begin{bmatrix} 0\,1\,0\,1 \\ 1\,0\,1\,0 \\ 0\,1\,0\,1 \\ 1\,0\,1\,0 \end{bmatrix}$$

Note that in D_1, the diagonal consists of 2×2 blocks of zeros and ones. This means that along the x-axis, the distance to cases in the same class is zero and the distance to cases in the other class is 1. Along the x-axis, the cases are perfectly separated. Along y-axis, they are not separated well. In this case, the ideal weight vector is $(1, 0)$, which would make the final distance matrix $D_w = D_{(1,0)} = D_1$.

We perform a single weight update as shown in Algorithm 1, starting with weights $(0.5, 0.5)$. The incrementally updated weight vector for each case is shown. We use the training set $T = Tr$ updating for all cases. The indexes of the nearest hit and miss cases are $H = (2, 1, 4, 3)$ and $M = (3, 4, 1, 2)$, where H_i denotes the hit for the ith case. The weight vector changes after considering each case as follows:

Case 1 : 0.714 0.286
Case 2 : 0.959 0.041
Case 3 : 0.999 0.001
Case 4 : 0.999 0.001

The weight obtained after Case 4 is the new weight vector. Performing a second iteration should result in the same weight vector; thus, the weights have converged. Note that for the last two updates, the computed weights were not in $[0, 1]$. After each update, we normalize the weights to be in $[0, 1]$ and sum to 1. The learned weights converge to the weights we expected.

3.4 Case Base Maintenance

In case-based reasoning, the case base often contains irrelevant or near-duplicate cases, which degrades performance. Case base maintenance methods organize the case base by removing such cases. This is typically accomplished using clustering, where the case base is partitioned into a set of clusters. Next, a prototype of each cluster is selected and becomes the representative of the cluster. Cases that have high distance from the prototype can be removed or at least assigned a low weight. Distance function learning is often used for

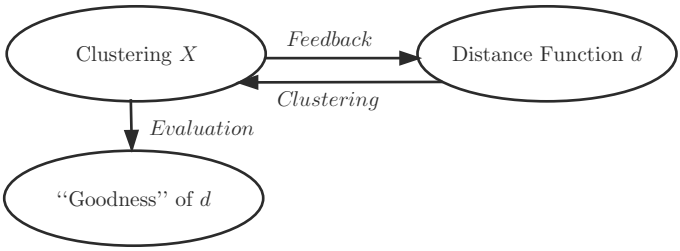


Fig. 3.2. Coevolving clusters and distance functions

case base maintenance. This creates a customized case base for the distance function.

If the case base has class labels, we seek a distance function such that the resulting clusters are more pure. Purity is defined as the average fraction of cases in a cluster that belong to a majority class. When the purity approaches 1.0, all cases in all clusters belong to a single class. The algorithm we discuss in this section, inside–outside weight updating, relies on two ideas, as depicted in Fig. 3.2. First, it uses clustering to evaluate the weighted distance function shown in (3.5). Second, it uses the local information from clusters to update the weights. We refer to this as a *coevolving* approach because the clustering and the distance function are learned together.

After the cases have been clustered, the purity of the obtained clusters is computed and is used to assess the quality of the current distance function. Any clustering algorithm can be used for this purpose; however, supervised clustering algorithms [18] that maximize cluster purity while keeping the number of clusters low are particularly useful for supervised distance function evaluation. More details on how clustering is exactly used for distance function evaluation are given in [17].

Inside–outside weight updating uses the average distance between the majority class members [1] of a cluster and the average distance between all members belonging to a cluster for the purpose of weight adjustment. More formally, let

$d_j(x, y)$ be the distance between x and y with respect to the jth attribute.

w_j be the current weight of the jth attribute.

σ_j be the average normalized distances for the cases that belong to the cluster with respect to d_j.

μ_j be the average normalized distances for the cases of the cluster that belong to the majority class with respect to d_j.

[1] If there is more than one most frequent class for a cluster, one of those classes is randomly selected to be "the" majority class of the cluster.

The weights, w_j, are adjusted with respect to a particular cluster:

$$w'_j = w_j + \alpha w_j(\sigma_j - \mu_j) \tag{3.8}$$

with $0 < \alpha \leq 1$ being the learning rate.

After a clustering[2] has been obtained with respect to a distance function the weights of the distance function are adjusted using Formula (3.8) iterating over the obtained clusters and the given set of attributes. It should also be noted that no weight adjustment is performed for clusters that are pure or for clusters that only contain a single case.

The weight adjustment formula (3.8) gives each cluster the same degree of importance when modifying the weights. Suppose we had two clusters, one with 10 majority cases and 5 minority cases and the other with 20 majority and 10 minority cases, with both clusters having identical average distances and average majority class distances with respect to a feature, the weight would have identical increases (decreases) for the two clusters. This somehow violates common sense; more efforts should be allocated to remove 10 minority cases from a cluster of size 30 than removing 5 members of a cluster that only contains 15 cases. Therefore, we add a factor λ to the weight adjustment heuristic that makes weight adjustment somewhat proportional to the number of minority cases in a cluster. Our weight adjustment formula therefore becomes

$$w'_j = w_j + \alpha\lambda w_j(\sigma_j - \mu_j) \tag{3.9}$$

with λ being defined as the number of minority cases in the cluster over the average number of minority cases for all clusters. Because the approach tends to move majority cases to the center of a cluster and nonmajority cases away from the center, it is called inside–outside weight updating.

This update rule is similar to that of the Relief algorithm as discussed in Sect. 3.3. Like this method, Relief updates the weight to bring same-class cases closer together and different-class cases further apart. Unlike this method, Relief uses simple nearest-neighbor queries instead of whole-cluster information. It does not take advantage of information from both the class labels or the result of the clustering. For example, (3.7) puts more emphasis on the cases in the same class compared to different classes. In the context of clusters, this expression is similar to the μ term from (3.9).

Example 5. Assume we have a cluster that contains six cases numbered 1 through 6 with cases 1, 2, 3 belonging to the majority class. Furthermore, we assume there are three attributes with three associated weights (w_1, w_2, w_3) which are assumed to be equal initially $(w_1 = w_2 = w_3 = \frac{1}{3})$, and the decomposed distance matrices D_1, D_2, and D_3 with respect to the three attributes are given below:

[2] Clusters are assumed to be disjoint.

$$D_1 = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 3 \\ & 0 & 1 & 2 & 3 & 1 \\ & & 0 & 1 & 2 & 2 \\ & & & 0 & 1 & 3 \\ & & & & 0 & 1 \\ & & & & & 0 \end{bmatrix} \quad D_2 = \begin{bmatrix} 0 & 1 & 1 & 1 & 5 & 1 \\ & 0 & 1 & 1 & 5 & 1 \\ & & 0 & 1 & 5 & 5 \\ & & & 0 & 5 & 1 \\ & & & & 0 & 5 \\ & & & & & 0 \end{bmatrix} \quad D_3 = \begin{bmatrix} 0 & 3 & 3 & 2 & 2 & 3 \\ & 0 & 3 & 2 & 2 & 3 \\ & & 0 & 2 & 2 & 2 \\ & & & 0 & 1 & 3 \\ & & & & 0 & 1 \\ & & & & & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 1.67 & 2 & 2 & 3.67 & 2.33 \\ & 0 & 1.67 & 1.67 & 3.33 & 1.67 \\ & & 0 & 1.33 & 3 & 3 \\ & & & 0 & 2.33 & 2.33 \\ & & & & 0 & 2.33 \\ & & & & & 0 \end{bmatrix}$$

The case distance matrix D is next computed using

$$D = w_1 D_1 + w_2 D_2 + w_3 D_3$$

First, the average cluster and average inter-majority case distances for each attribute have to be computed; we obtain $\sigma_1 = 2$, $\mu_1 = 1.7$; $\sigma_2 = 2.6$, $\mu_2 = 1$; $\sigma_3 = 2.3$, $\mu_3 = 2.5$. The average distance and the average majority cases distance within the cluster with respect to d are $\sigma = 2.29$, $\mu = 1.78$. Assuming $\alpha = 0.2$, we obtain the new weights:

$$\begin{aligned} w' &= \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right) * (1.06, 1.32, 0.953) \\ &= (0.353, 0.440, 0.318) \end{aligned}$$

where $*$ denotes the element-wise multiplication. After the weights have been adjusted for the cluster, the following new case distance matrix D is obtained:

$$D = \begin{bmatrix} 0 & 1.57 & 1.89 & 1.92 & 3.82 & 2.21 \\ & 0 & 1.57 & 1.60 & 3.50 & 1.57 \\ & & 0 & 1.29 & 3.19 & 3.19 \\ & & & 0 & 2.58 & 2.21 \\ & & & & 0 & 2.58 \\ & & & & & 0 \end{bmatrix}$$

The average inter-case distances have changed to $\sigma = 2.31$, $\mu = 1.64$. As we can see, the cases belonging to the majority class have moved closer to each other (the average majority class case distance dropped by 0.14 from 1.78), whereas the average distances of all cases belonging to the cluster increased very slightly, which implies that the distances involving majority cases (involving cases 1, 2, and 3 in this case) have decreased, as intended.

3.5 Case-Based Retrieval

In case-based retrieval, cases are retrieved and ranked according to their distance to a query. Unlike classification and maintenance methods, the cases are assumed not to have a class label. Offline training of a distance function is therefore not possible. A user's query, at runtime, partitions the set of cases

into two classes: relevant and nonrelevant. Distance function learning uses this feedback to improve the recall and precision of subsequent retrievals. Since a new distance function is learned for each query, the learning process must be very fast, even if not very accurate.

Learning a distance function in response to a user's feedback is known as relevance feedback. One of the earliest relevance feedback algorithms was designed for text retrieval [37]. As in other fields, a case t is represented as a vector of word features:

$$\mathbf{t} = (t_1, \ldots, t_n)$$
$$t_j \geq 0$$

where each t_j is the value for the jth word from a set of all n words that occur in the case base.

Although cases are represented by feature vectors much like the other fields we have discussed, the values for the features have a special meaning. This permits several assumptions in these case bases that are otherwise not justified for the other domains we have considered. The most common form is the term-frequency inverse case frequency (TFIDF) where each term has the following form:

$$t_i = \begin{cases} \text{FREQ}_i \left(1 - \frac{\log_2 \text{DOCFREQ}_j}{\log_2 m}\right) & \text{word}_j \in t \\ 0 & \text{word}_j \notin t \end{cases}$$

where DOCFREQ_j is the number of cases that contain the jth word, FREQ_j is the number of times the jth word occurs in the case t, and m is the number of cases in the case base [39].

3.5.1 Distance Function Learning

The most common distance function learning algorithm for case-based retrieval using relevance feedback relies on some assumptions about relevant cases and the words they contain. An analysis of the distribution of words in text cases has shown the following [39]:

1. Relevant cases depend on a small set of relevant words (assumed to be in the query)
2. Relevant words are rare across all cases (basis for the TFIDF feature representation)
3. Relevant words are consistent across relevant cases

The most frequent words in a case base tend to be irrelevant to any particular topic, although most of these are removed during preprocessing. Within the relatively small set of relevant cases, the cases are relevant because they

contain these relevant words. As a result, the relevant words are expected to occur consistently and more frequently in the set of relevant cases. Note that the query does not always contain all relevant words, as the words may have synonyms. In this case, the query may miss cases that use different words to describe the same topic.

Since we assume that relevant cases are characterized by the consistent presence of relevant words, distance function learning methods weight words by their consistency within the set of relevant cases. A common measure for inconsistency is the standard deviation of the word frequencies in the relevant cases. The weights are inversely proportional to the standard deviation:

$$w_i = \begin{cases} \frac{1}{\sigma_{i,r}} & \sigma_{i,r} \neq 0 \\ w_0 & \sigma_{i,r} = 0 \end{cases}$$

where $\sigma_{i,r}$ is the standard deviation of word i with respect to the relevant cases r and w_0 is a fixed weight when the standard deviation is close to 0. For words that consistently appear (and do not appear) in the relevant cases, the standard deviation will be low and their weight will be high. For words that do not appear consistently, the standard deviation will be high and their weight will be low.

Despite the apparent simplicity of the weight update strategy, it has been shown to be effective for text retrieval [8,37,39]. The main reason for its good performance is that it has access to the relevant cases, rather than having to predict class labels. In pattern classification and clustering, the relevance of a new case (class label) is not known at runtime, the classification process is not interactive. Information retrieval is an interactive process in which the task is slightly different than classification. Instead of asking "What is the best class of this case?" the question is "Given the relevant cases, what other cases are similar to these?" By providing cases of relevant cases interactively, the user provides significantly more information to the retriever than is available to the classifier. Since distance functions are individualized and different from each other, each distance function determines what is relevant for a particular user but not what is relevant to all users.

Evaluating the performance of these algorithms is difficult in practice. The retrieval process is evaluated in terms of precision (percent of the returned cases that were relevant) and recall (percent of relevant cases that were returned). Ideally, both precision and recall should be high. For relevance feedback algorithms, users are typically not available during experiments. A common practice is to simulate a user's feedback by assigning a class label to each case. Cases in the same class as the query are then considered relevant.

Example 6. Suppose we have a corpus (case base) of five cases with the following most frequently occurring words:[3]

[3] This example is intended to demonstrate the algorithm and not any political opinion.

1. George near a bush
2. George Herbert Walker Bush
3. George Clinton
4. The President
5. President Bush

Each case is the set of most frequently occurring words in a text document. Following the standard text preprocessing procedure, single letter words and articles ("the," "a") are removed and capitalization is ignored. The vector representation of the case base is

$$t = \begin{bmatrix} \text{bush} & \text{clinton} & \text{george} & \text{herbert} & \text{near} & \text{president} & \text{walker} \\ 0.5 & 0 & 0.8 & 0 & 0.2 & 0 & 0 \\ 0.6 & 0 & 0.6 & 0.6 & 0 & 0 & 1 \\ 0 & 0.7 & 0.6 & 0 & 0 & 0 & 0 \\ 0.1 & 0 & 0.01 & 0 & 0 & 0.8 & 0 \\ 0.6 & 0 & 0.01 & 0 & 0 & 0.9 & 0 \end{bmatrix}$$

The query of "President George W. Bush" results in the vector q:

$$q = (1, 0, 1, 0, 0, 1, 0)$$

With equal weights, cases 5, 1, 2, 4, 3 are returned in that order. The user now selects cases 4 and 5 as relevant. The normalized weight vector is then w':

$$w' = (0.17, 0.0, 0.0, 0.0, 0.0, 0.83, 0.0)$$

This assigns the highest weights to "President" and "Bush." The weight for "Bush" is less than that for "President" because "Bush" appears inconsistently across the relevant cases, in case 5 but not 4. The word "President," however, appears in both cases. The same query with the learned weights returns the cases 5, 4, 2, 1, 3 in that order. After relevance feedback, the top two cases were those marked as relevant.

3.6 Visualizing Proximity of Cases

Because distance functions are embedded in different algorithms (classification, clustering, and retrieval), it can be difficult to evaluate the performance of a distance function learning algorithm. Both to provide an intuitive understanding of distance functions and as a qualitative evaluation, we discuss several visualization methods for distance functions. They show how learning a distance function changes the spatial relationship among the cases, and try to visualize how good distance functions can be distinguished from bad ones.

3.6.1 Voronoi Diagrams

A Voronoi diagram partitions cases in a case base into mutually exclusive regions called Voronoi cells (See [33] for more details). Each cell c contains exactly one case x from the case base $Y \subseteq X$. The cell is defined such that the case x is the nearest case in the case base to all cases in the cell. Cases along the border are equally distance to the cases in the cells.

The weights change the shape of the cells in the Voronoi diagram. Figure 3.3 shows two Voronoi diagrams of a case base with two classes. On the left the cells appear to be wider but compressed vertically. This is because our original case base is skewed along one dimension. We change the weights to place more "emphasis" on the x-axis such that its weight is larger than that of the y-axis. As a result, the cells on the right diagram are stretched along the vertical dimension and compressed along the horizontal axis. As the weight for x increases towards 1, the cells become vertical line segments.

3.6.2 Multidimensional Scaling

In the 2D case, we saw that a slight change of the weights significantly changed the neighborhood of the cases. In the Voronoi diagram, this is easily seen as changing the shape of the Voronoi cells. In higher dimensions, however, computing the Voronoi diagram is computationally expensive.

Multidimensional scaling (MDS) (See [6] for more details) finds a p-dimension representation for n-dimension cases, where $1 \leq p < n$. For visualization, we assume that $p = 2$. A key benefit of MDS for visualization is that it preserves the distance between cases such that

$$d(\phi(x), \phi(y)) \approx d(x, y)$$

where x, y are cases in the original case base and $\phi : \mathbb{R}^n \to \mathbb{R}^p$ projects the input cases to the lower dimension.

Most MDS algorithm operate only on the distance matrix of the cases. In classical multidimensional scaling, the projection is a decomposition of the distance matrix. Since the purpose of distance function learning is to change the distance function and thus the distance matrix, cases change their spatial relationship as the weights change. For many MDS algorithms, the change in position is so drastic that it is hard to compare the two figures before and after changing the weights.

3.6.3 Distance Matrix Image

Although MDS can give us a nice visualization of the cases in 2D, much visual information is lost when moving from one set of weights to another. This is because the cases tend to move in seemingly unpredictable ways. The distance matrix image does not suffer from these problems. The main disadvantage, however, is that it can often be hard to see small intensity changes.

To form the distance matrix image, each entry becomes a block of pixels in an image. The color of each block is proportional to the distance. A common color map is a linear gradient from black to white. In this color map, the brighter the color, the larger the distance. Each row i in the image corresponds to $D_{i,j}$ for all columns j. As the weights change in the distance function, the color of the pixels will change.

To facilitate the qualitative meaning of the image, adjacent rows should correspond to cases in the same class. Grouping by class allows us to observe changes in the structural properties of the distance function. Blocks along the diagonal represent the same classes. Off-diagonal blocks represent distance between pairs of classes, and classes that are well separated will have bright colors for these blocks. Cohesive classes in which the cases are all close to each other will have dark colors in the blocks along the diagonal. We typically assume that classes are cohesive, but this may not be true in general.

3.6.4 Examples of Learned Distance Functions

We show two example case bases before and after changing the weights. For the 2D case base, we demonstrate all three visualization methods. For the 9D case base, we demonstrate only the last two methods. In both cases, the objective of learning the objective function is to improve classification performance. The objective function should make cases in the same class closer and cases in different classes further apart.

Example: 2D Random Case Base

Figure 3.3 shows about 20 cases each from two 2D Gaussian distributions with means $\mu_1 = (0.8, 1)$, $\mu_2 = (1, 1)$, and covariance $\Sigma_1 = \Sigma_2 = \begin{bmatrix} 0.006 & 0 \\ 0 & 0.04 \end{bmatrix}$. In Figs. 3.3, 3.4, and 3.5, the weights are

$$w_u = (0.5, 0.5)$$
$$w_r = (0.7, 0.3)$$

where w_r has unequal weights to simulate learning the distance function.

Voronoi Diagrams

Figure 3.3 shows Voronoi diagrams for the cases with equal and unequal weights. The diagram with unequal weights has been stretched vertically but compressed along the horizontal axis. It may be unclear why the cells are taller when the vertical weight is smaller. This is because a smaller weight decreases distance along that dimension. As a result, more cases are closer to the cases in the cells along that dimension. In contrast, horizontal distance has increased leaving fewer cases close in the cells.

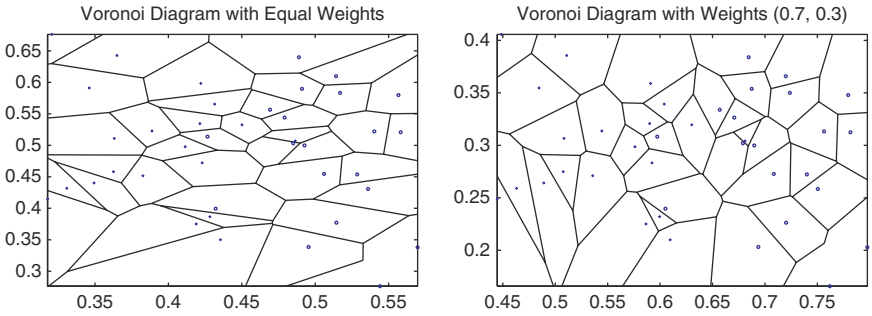


Fig. 3.3. Voronoi diagrams for (*left*) equal weights and (*right*) weights $(0.7, 0.3)$. The scales are different in the figures because the algorithm that computes the Voronoi diagram only uses unweighted Euclidean distance

The Voronoi diagram indicates that the distance function meets our objectives. The separation between μ_1 and μ_2 is greatest along the horizontal component because $|\mu_1 - \mu_2| = (0.2, 0)$. The weights increase the horizontal distance causing the means to be further apart. The result is that the distance function can potentially improve classification performance.

Multidimensional Scaling

Figure 3.4 shows the result of multidimensional scaling with equal weights and unequal weights. Although the original case base was 2D, with unequal weights, the position of the cases and thus their distance has changed significantly compared to the distance with equal weights.

The figures show that with unequal weights, the grouping of the cases improves. As a whole, the classes appear to be further separated from each other in Fig. 3.4 (right). Within each class, the cases appear to be closer together.

Distance Matrix Image

Figure 3.5 shows the distance matrix image for the cases with equal and unequal weights. Since the cases are grouped by class, the lower-left and upper-right quadrants of the figure denote cases in the same class. With unequal weights, these regions are generally darker. This means that the distance between cases in the same class has decreased overall. The lower-right and upper-left quadrants of the figure denote the distance between cases in different classes. With unequal weights, these quadrants are generally brighter. This means that the distance between cases in different classes has increased overall.

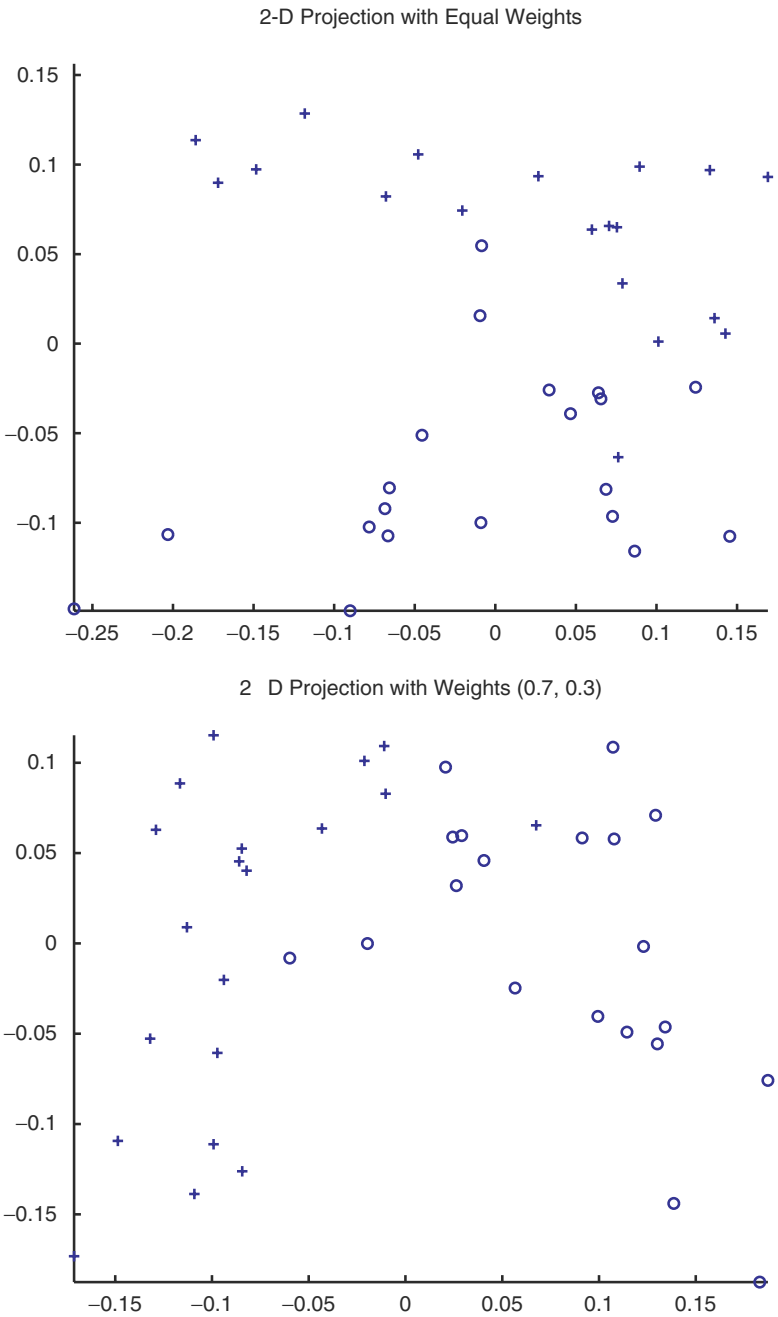


Fig. 3.4. Multidimensional scaling of the 2D case base with (*top*) equal weights in the original case base and (*bottom*) weights $(0.7, 0.3)$

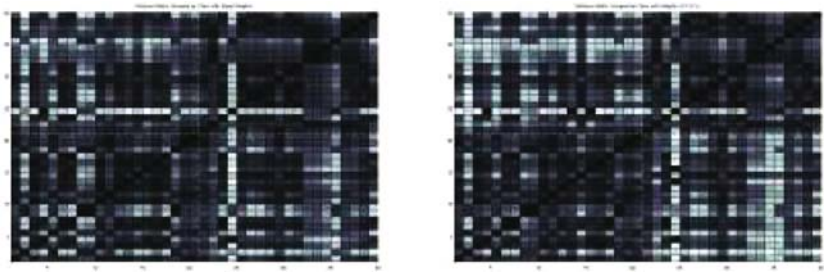


Fig. 3.5. Distance matrix images for the 2D case base with (*left*) equal weights and (*right*) weights $(0.7, 0.3)$.

Example: Nine-Dimensional Case Base

With a 2D case base, the Voronoi diagrams were illustrative. Unfortunately, for more than three dimensions the Voronoi diagram is expensive to compute and difficult to display. For higher-dimensional case bases we use only the multidimensional scaling and the distance matrix images.

The case base has nine features with 214 cases from seven classes [4]. We demonstrate the effect of weights with two weight vectors:

$$w_u = \frac{1}{9}$$
$$w_r = (0.0646, 0.0936, 0.4283, 0.1275, 0.0466, 0.0340, 0.0826, 0.1121, 0.0109)$$

where w_u consists of uniform weights and w_r is the weight vector learned by Relief-F algorithm [30]. As discussed in Sect. 3.3, Relief is designed to find weights that increase the distance between cases of different classes and decrease the distance between cases of the same class.

Multidimensional Scaling

Figure 3.6 shows the multidimensional scaling of the case base with equal weights and with Relief weights. We see that the Relief algorithm works well on this case base. Unlike the 2D case, the cases have shifted significantly with the different weights. In general, the cases are further apart with the Relief weights because the scale of the figures is different. Cases in the same class are grouped together, which is desirable in classification problems. The clusters are better separated from clusters of cases belonging to another class. In general, the figure shows that cases are more tightly grouped within the same class and these groups are better separated from each other.

Distance Matrix Image

Figure 3.7 shows images of the distance matrix with equal weights and Relief weights. These figures show that the Relief weights increase the distance

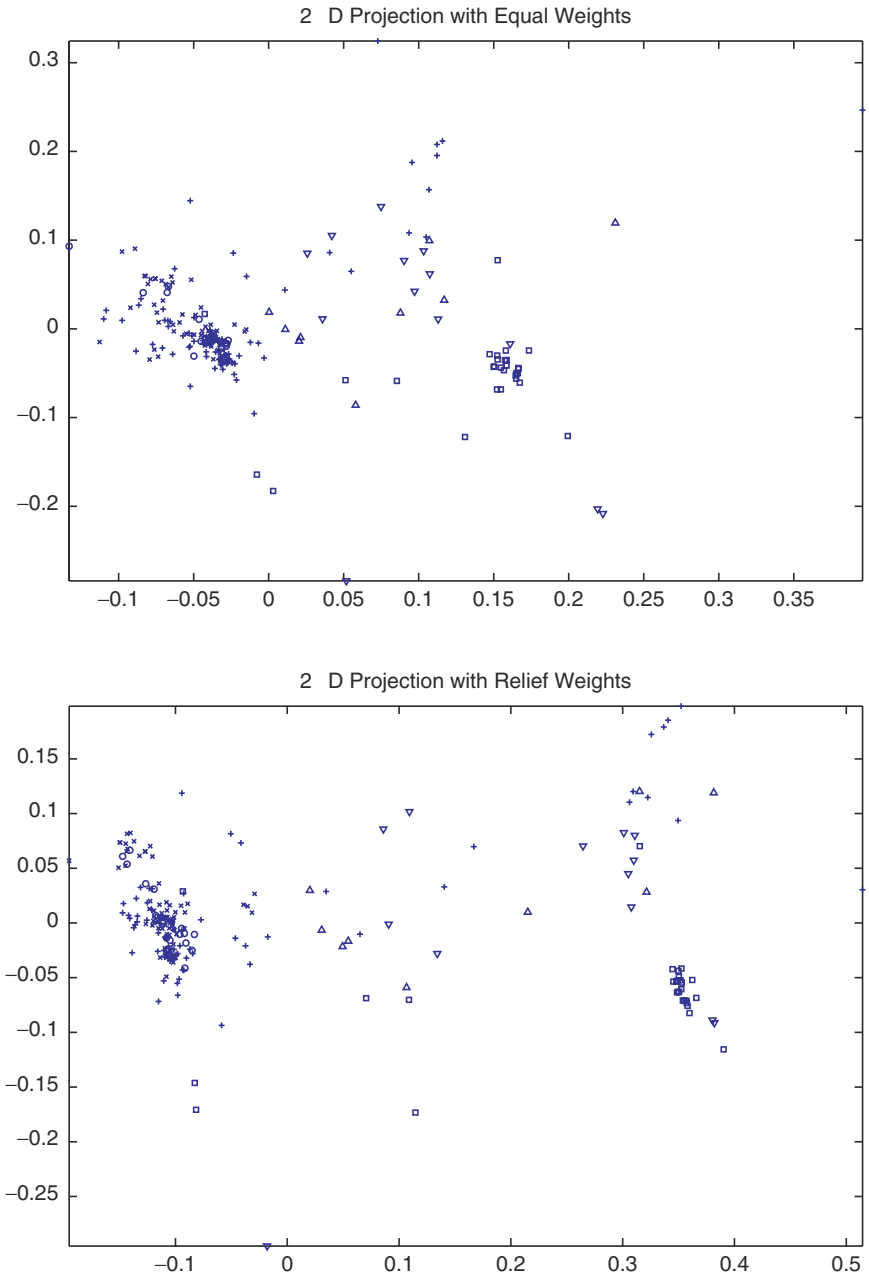


Fig. 3.6. Multidimensional scaling of the glass case base with (*top*) equal weights and (*bottom*) Relief weights [30]

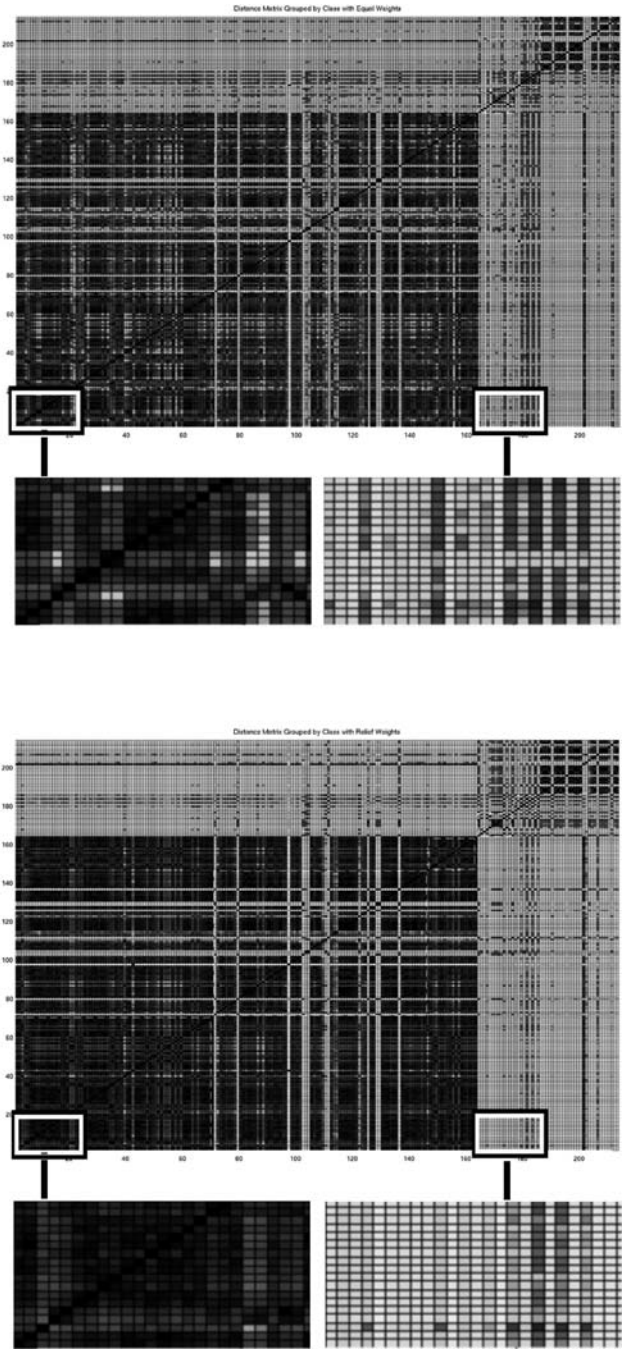


Fig. 3.7. Distance matrix images for the 9D case base with (*top*) equal weights and (*bottom*) Relief weights. The enlarged portions illustrate the difference between intraclass distances (*left*) and interclass distances (*right*)

between cases of different classes but decrease the distance between cases of the same class. The enlarged portions of the figure highlight the difference between the distance functions. With equal weights, the distance between cases in the same class decreases, causing the image to appear darker. With Relief weights, the distance between these cases and those of different classes increases, causing the image to appear brighter. Among the rest of the cases, the blocks along the diagonal appear darker with Relief weights. This means that cases in the same class are closer to each other with Relief weights. The off-diagonal blocks, particularly on the top and right, are brighter with Relief weights.

The distance matrix images also reveal some distance structure across the classes. If the classes were separated well, most of the distance matrix would be brightly colored. Only the regions along the diagonal would be dark. Since the majority of the distance values are small, this means that the three classes in the upper-right are, as a whole, very far from the rest of the cases. The Relief weights clarify this fact.

3.7 Related Work

In this section, we survey other distance function learning work in case-based reasoning and machine learning. The work is grouped by the method and objective. Our survey is broad in scope and touches several different fields of research, each placing different requirements on distance functions. Although we do not claim that our survey is complete, we hope it serves as a starting point for newcomers to the field.

3.7.1 Statistical Methods

Statistical methods make use of either simple statistical models or models of the probability distribution. The model is used to derive weights and usually depends on distributional assumptions about the case base.

Correlation

The correlation $\rho(X, Y)$ between two random variables X and Y is defined as

$$\rho(X, Y) = \frac{\Sigma_{X,Y}}{\sqrt{\sigma_X^2 \sigma_Y^2}}$$

where σ^2 is the variance of each random variable and Σ is the covariance between the random variables. The correlation has a range of $-1 \leq \rho(X, Y) \leq 1$. When $\rho=1$, the variables are said to be positively correlated, which means that as X increases, Y increases. When $\rho = -1$, the variables are negatively correlated, meaning that as X increases, Y increases. In distance function

learning, we are interested in the magnitude of the correlation, which indicates the degree to which a feature is correlated (positive or negatively) with the target value. Both kinds of correlation mean that the feature should have a high weight.

Correlation is used for both selecting features and assigning weights. Yu and Liu select features that are highly correlated with the class labels but not correlated with each other [51]. This reduces redundant features because features that are highly correlated may be redundant. Doulamis and Doulamis set feature weights proportional to the correlation between the features values and a user-defined relevance measures, as in information retrieval [15]. Pan et al. find a highly correlated subset of features of a high-dimensional projection of the case base [34]. Projecting features into a high-dimensional space could introduce many irrelevant or redundant features. To eliminate redundant features, we compute a kernel matrix and perform an eigenvalue decomposition as in kernel PCA. The k largest eigenvalues correspond to the most important features. Weights for this subset of features are proportional to the correlation coefficient with respect to the target.

Variance

As we discussed in Sect. 3.5, many relevance feedback methods use the variance of features in the case base because it is easy to compute. It is especially useful in interactive case-based systems, because users can specify which cases are relevant. Kohlmaier et al. assign weights to a feature based on the degree to which it increases variance in the computed similarity values [29]. The variance in the similarity function is a good indicator of different cases. In addition to providing the single best response, a good case result should return a variety of different, but related, cases. We consider adding a single feature to the set of features. If this new feature increases the variance of the similarity function, it is believed to be a good starting point for obtaining good feedback from the users. If the variance is low with this feature, most of the cases are similar based on this feature, so it may not help the user find a good solution. The variance of similarity values can be used either alone or combined with weights. Results show that the method performs better than weights alone on several case-based retrieval tasks [40].

Expectation Maximization

Expectation maximization (EM) approaches iterative solve an optimization problem where the objective is to maximize the likelihood of the case base given the new model. In distance function learning, the model is the set of weights. The likelihood is the probability of finding the existing case base given the current weights. The EM algorithm has two basic steps. In the maximization step, we find those weights that best explain the case base. In the expectation step, we recompute the likelihood function, which changes

when the weights change. The EM algorithm maximizes weights, computes the expected case base given the weights, and then repeats until the weights converge.

These methods are common in classification and clustering. In the maximum likelihood approach, we seek a set of weights that best explains the case base. If we have a set of pair-wise constraints, such as knowing that the two cases should not be considered similar, we can find weights that maximize the likelihood of separating cases [50]. Given the weights, we determine the degree to which the pair-wise constraints are violated and update our likelihood function. We can again find weights that maximize the likelihood. Huang et al. apply another EM-type algorithm to find weights that induce a good clustering, a maximum-likelihood clustering. Each cluster is modeled by a multivariate Gaussian distribution. Each case has a probability of being a member of each cluster. The weights are used to compute the membership probabilities for each case. The EM approach is to obtain a new clustering given the weights and then find the weights that improve the membership probabilities [24].

3.7.2 Changing the Set of Features

Often the original set of features is not adequate for the distance function. We would like to evaluate a feature, indicating its usefulness with respect to the class labels. When there are many features, some are irrelevant or redundant. Finding a suitable subset or subspace of the features, these features can improve the computational efficiency. When the features do not adequately express the target value, we can construct new features that are more expressive.

Feature Evaluation

In Sect. 3.3, we discussed the popular Relief method. Many other feature evaluation algorithms are commonly used to find weights for distance functions. Most of these were designed, like Relief, for classification problems and to compare the class distribution considering the different values of nominal features. Information gain measures the difference in entropy with respect to the class labels [16]. The case base is split into partitions with respect to a particular feature such that all cases in the partition have the same value for the feature. The entropy with respect to the class labels is calculated in each partition. The information gain is the difference between the initial entropy, without the partitions, and the average entropy after the partitioning.

As with the Relief algorithm, the output of a feature evaluation method can be used for weights in a distance function. In classification, the information gain increases weights for features that are similar to the class label. Features that are distributed like the class label have high weights as do those features with many values [14]. Text features, particularly in information retrieval,

lend themselves to this method. A relevant word would only be in the set of relevant cases, so we would expect it to have high information gain. If the word is not relevant the class labels would tend to be distributed similarly in each split, so the information gain would be small [49].

Extending the concept of feature weights to individual feature values is also common in case-based reasoning literature. Such a local weight could, for example, place more emphasis on the difference between the values "red" and "blue" than on the difference between "red" and "pink". Nunez et al. compared several entropy-based weighting methods for local weighting [32]. They showed that the local weight approach can be useful. However, having more weights to control makes the learning process more difficult. As a result, more complex search methods such as genetic algorithms have to be used [25].

Decreasing the Number of Features

As we have seen, feature selection seeks an optimal subset of the initial set of features. In dimension reduction, we also seek a reduced set of features, but we consider combinations of features. As shown in our visualization cases in Sect. 3.6, multidimensional scaling can reduce a high-dimensional case base to a 2D case base that preserves clusters. There are three common methods for dimension reduction: principal component analysis, linear discriminant analysis, and multidimensional scaling.

Principal Component Analysis

The principal components of a case base are the directions (linear combination of features) that indicate the variance in the cases. They are the eigenvectors of the covariance matrix that correspond to the largest eigenvalues. The principal components become the new features. The covariance matrix allows to calculate the Mahalanobis distance, which projects the cases onto the inverse of the covariance matrix. The projected cases is "corrected" for the covariance such that an unweighted distance function can be used. The objective of these distance function learning algorithms is to find the best projection. For example, the projection should map the cases into well-separated classes [5,19].

Linear Discriminant Analysis

Rather than requiring a distance function to consist of good features, one can require that it lead to good classification results. Since predicting accuracy is computationally expensive, an alternative is to maximize the margin of separation between cases. Since the separation depends on the learned distance function, the problem is to find a distance that leads to the greatest separation between cases in different classes. An early algorithm for this purpose is Linear Discriminant Analysis (LDA). A local approach to LDA finds the discriminant among a small neighborhood of cases. The linear projection matrix that leads to the best discriminant is chosen for each case [22].

Multidimensional Scaling

From a geometric perspective, these covariance approaches reshape the cases to improve separation. The dimensionality of the projected cases is the same as the original cases. As we have seen in Figs. 3.4 and 3.6, projecting cases onto fewer dimensions can yield insight into the distances. Much work has shown that they are quantitatively effective as well. A desirable property of multidimensional scaling is that it should preserves distances in the projected space. Rather than preserving all the original distances, the projection can increase the distance between cases in different classes, leading to a better separation of classes in the lower-dimensional space [52].

Other projection methods utilize different aspects of relatedness such as knowledge of unwanted variance between cases of different classes [23] and the structure of local neighborhoods [20].

Increasing the Number of Features

Feature Construction

Often the original set of features for a case base are unable to represent the concept. For example, if the true target value is a nonlinear function of the features such as $x^2 + xy$, then it would be difficult for any weight vector to approach the function. A common solution is to construct a new, larger set of features. The similarity function and weights are then computed in this high-dimensional feature space. Examples include the set of polynomials of degree 2, yielding features such as $(x_1, x_2, x_1^2, x_2^2, x_1 x_2)$. This feature construction method can introduce significant redundancy among the features. As a result, dimension reduction methods, like those discussed earlier, can be used. A specialized version of PCA, called kernel PCA, is ideally suited for this problem [34]. As the original number of features increases, feature construction like this tends to be very expensive, as $O(n^2)$ features have to be constructed. Another technique is to add new, derived features. An example is to find a set of weights for a subset of features and then use the weighted combination of feature values as a new feature [36]. The new feature can either replace or augment the existing set of features. It is particularly useful with image cases, when the individual low-level features are, by themselves, not good predictors of the class.

Kernels

In kernel-based machines, cases are represented as a vector of similarity values. Each case consists of its similarity to all other cases in the case base. A kernel function $k(x_1, x_2)$ defines the similarity between two cases x_1 and x_2. The kernel matrix is constructed from the original case base as follows. For a case base of size n, the kernel matrix has dimension $n \times n$. Each entry $K_{i,j}$ is the value of the kernel for two cases $K_{i,j} = k(x_i, x_j)$. Using the so-called kernel

trick, any algorithm that operates on case bases with a dot product can be modified to use a kernel function. This kernelization allows many existing algorithms to be extended with kernels. Techniques that can be kernelized are the support vector machines, principal component analysis, and distance based algorithms [3, 41]. In kernel PCA, we compute a reduced-dimension version of the kernel matrix. The weights for this new feature vector indicate which cases contribute most to the overall covariance of the kernel matrix. As with traditional PCA, the kernel version improves the performance of algorithms when using the reduced set of features [34].

Recent work has established a relationship between kernels and distance functions [3, 41]. In particular, the work shows that good distance functions make for good kernels [3]. As a result, many researchers have applied distance function learning methods, like those discussed in this chapter, to learn kernel functions. As in distance function learning, feature evaluation methods like Relief can be used to find weights for kernel-based machines [9]. Weights can be learned for a parameterized kernel function in much the same way they are learned for distance functions [26]. Recent work shows that good distance function make for good kernels [3].

3.7.3 Extracting Weights from Learned Models

A common approach to assess the importance of features is to apply an algorithm that generates weights as part of the model. The weights are then extracted from the model and used as weights for the distance function or to select features.

Classifiers

A popular classifier that computes weights is the Logistic regression algorithm. This algorithm assumes that the probability of a case x belonging to a class y is

$$Pr(y \mid x, w) = \frac{1}{1 + e^{-w^T x}}$$

where w is a vector weight and includes a bias. The algorithm is trained using a gradient descent approach. The learned weights are then used for feature selection and weights in a distance function.

Arshadi and Jurisica have applied logistic regression to case-based classification of microarray cases [1]. Their objective was to select relevant features from a very high dimensional case base. They combine several classifiers in an ensemble. The classification approach is to retrieve several cases from the case base with the learned weights and then compute the majority class label. Their results show a significant improvement in accuracy. Wilson and Leake use this method to maintain the case base through clustering [48]. By clustering and

then learning a set of weights, cases that are very distant from the prototype of the cluster are removed as irrelevant. Features with low weights are also removed as irrelevant. Their results showed that this lead to an improvement of classification accuracy.

Support vector machines also learn weights, classifying cases based on a separating hyperplane as follows:

$$f(x) = \text{sign}(w^T x + b)$$

where w is a weight vector and b is a bias. The learned hyperplane is typically one that separates cases into two classes by the largest possible margin. The decision function $f(x)$ is the distance between a case and the separating hyperplane, so this function can itself be used as a distance function. Alternatively, the weights are extracted and used as weights for a distance function [13].

Relevance Feedback

The relevance feedback methods like those we have previously discussed use relevant cases as the basis for weights. Their objective is to find weights that best separate relevant from nonrelevant cases. Applied both in text retrieval [8, 37, 39] and image retrieval [38], these methods are quite popular. We can also cluster the cases based on relevance and then find a distance function that separates the clusters [27]

3.7.4 Local Search Methods

Local search methods are popular for distance function learning. The most common of these are iterative methods such as hill-climbing. Here, an initial weight vector is updated until the objective function converges, e.g., to the peak of a hill in the objective function. Updates are computed either by the gradient of the objective function or with heuristics. Relief, as we discussed in Sect. 3.3, is a local search method.

Extensions to Relief

In the Relief algorithm, a weight is the degree to which a single feature can be used to predict the class label [28]. In practice this technique works well and has inspired several extensions. The most common extension, Relief-F, extends Relief from two classes to several classes, which allows for greater applicability and widespread use [30]. Rather than learning a single weight, we can find a pair of weights for each class: one for the nearest hit and miss cases [7]. Although widely used as a batch learning algorithm, a recent iterative version of Relief achieves comparable accuracy as an online learner and supports removing outliers [45].

Using Gradients

Gradient-based hill climbing methods, known as gradient descent methods, are the most common form of hill climbing algorithm. For distance function learning, these methods define an objective function in terms of weights and then compute its gradient. A common approach used in case-based retrieval is to find weights that compute an optimal ranking of the cases. For example, given a user's feedback of known ranks for a set of cases, weights can be learned that match the ranks. The error function is simply the squared difference between the known ranks and predicted ranks. This is a continuous, differentiable function, which lends itself to gradient-based methods [31, 43]. Coyle and Cunningham minimize the difference between a user's ranking of a set of retrieved cases versus those computed with uniform weights. The idea is to make the similarity with weights as different from uniform weights as possible. By saving these weights for individual users, a customized case base is created [11]. With a similarly objective function, Shiu et al. incorporate learned weights as a first step in their case base maintenance process [42]. Tsang et al. find weights that induce a good clustering to edit the case base. The objective is to improve cluster metrics such as intracluster distance (tightness of the cluster) [46]. The objective function for optimization minimizes the difference in the objective with learned weights vs. the set of uniform weights.

Nongradient Methods

In Sect. 3.4, we examined a nongradient approach that optimizes class purity [17]. Here the weights are changed along the single attribute that improves the purity the most. A subspace projection method, like hill climbing, updates weights along a predefined direction as long as the objective function improves [21]. This direction typically has components of several features.

3.7.5 Global Search Methods

Local search methods tend to converge to a point, known as a local optima. This local solution may not be the global, best solution. Global search methods are intended to find this global solution. Optimization methods are used when we know that there is only one optimal solution, typically because the objective function is convex. Most other search methods expand their search area with randomization.

Optimization

We can pose the distance function learning problem as one of the optimization. In general, we would like to find feature weights that minimize a global error function. Peleg and Meir find a subset of features that minimize the expected generalization error [35]. The objective is to minimize the margin cost for a

feature. Features that can adequately separate the cases have low margin cost. Features that are poor separators are not useful for classification. Weinberger et al. extend the optimization problem in the Relief algorithm. For Relief we considered the distance between the nearest hit and miss case for each case in the training set. Keeping the hit and misses fixed during optimization, the objective is to minimize the distance between cases in the same class and maximize the distance between cases in different classes [47].

Randomized Search

Genetic algorithms can directly search for weights and can evolve expressions for the distance function. Stahl and Gabel find weights for specific feature values. Each individual is a similarity table containing weights for pairs of feature values. The fitness function is the accuracy of the ranks generated with the weights [44]. Jarmulak et al. select features by evolving a feature bit mask. The fitness function is cross-validation accuracy with classification cases [25]. Using genetic programming, feature indexes and arithmetic operators are added to a syntax tree forming an arithmetic expression. The fitness of the expression is its estimated prediction accuracy [12].

Because objective functions over a clustering can be expensive to compute, we seek search methods that do not evaluate many (very similar) solutions. We can view the search for weights as a transition from one clustering, induced by the original weights, to another, induced by the changed weights. If weights lead to a good cluster, these weights, and those that led to them, form a path of weights that lead to a good clustering. By remembering this path, consisting of which choices were good and bad, the search can focus on good paths while avoiding bad ones. If the path leads to a clustering that has already been seen, the search can quickly switch to a different path rather than repeating work already done. Conceptually, this is an application of reinforcement learning to search, in which the best choices are remembered and reused [2].

3.8 Summary

The objective of distance function learning for supervised similarity assessment is to find a distance function that groups together cases belonging to the same class, while separating cases of different classes. We introduced a framework for parameterized distance functions, which depends on a vector of weights. We then provided detailed descriptions of algorithms used in three different fields within case-based reasoning: case-based classification, case base maintenance, and case-based retrieval. We showed how to visualize the difference between good and bad distance functions for high-dimensional case bases. Finally, we surveyed recent work in the literature.

Distance function learning is particularly suited to applications with a large number of dimensions, when it is difficult for us to determine which

features are the most important. In classification, the Relief algorithm finds features that separate cases from different classes. In case base maintenance, the inside–outside weight update concentrates cases of the same class into cohesive clusters. In case-based retrieval, relevance feedback adapts the distance function to a user's preference at run time.

Distance function learning is a very active research field, and it can benefit from a cross-fertilization of ideas from different fields. The algorithms discussed in this chapter originated in the fields of machine learning, data mining, and information retrieval. In the field of machine learning, recent work has established a relationship between kernels and distance functions. Distance function learning can be applied to obtain better kernels, and kernel methods can be used to derive good distance functions. This has been used in pattern classification. In data mining, distance function learning has found widespread use when users can specify an objective function to organize information. This suggests an interactive approach to unsupervised clustering in which users can explore a clustering by changing objective functions. In information retrieval, distance functions are tailored to individual users and their queries. As different modalities of information become available in addition to text (image, video, signals), distance function learning can be used to emphasize the relevant features with respect to a user's query. In all of these fields, distance function learning is the common thread helping us better assess the similarity between cases.

References

1. Niloofar Arshadi and Igor Jurisica. Maintaining case-based reasoning systems: A machine learning approach. In *Proc. 7th European Conf. on Adv. in Case-Based Reasoning*, LNAI 3155, pages 17–31, Madrid, Spain, September 2004.
2. Abraham Bagherjeiran, Christoph F. Eick, Chun-Sheng Chen, and Ricardo Vilalta. Adaptive clustering: Obtaining better clusters using feedback and past experience. In *Proc. 5th Int'l Conf. on Data Mining*, pages 565–568, Houston, TX, USA, November 2005. IEEE Computer Society.
3. Maria-Florina Balcan and Avrim Blum. On a theory of learning with similarity functions. In *Proc. 23rd Int'l Conf. on Machine Learning*, pages 73–80, 2006.
4. C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.
5. Leon Bobrowski and Magdalena Topczewska. Improving the k-NN classification with the Euclidean distance through linear data transformations. In Petra Perner, editor, *Adv. in Data Mining, Applications in Image Mining, Medicine and Biotechnology, Management and Environmental Control, and Telecommunications*, LNAI 3275, pages 23–32. Springer Verlag, 2004.
6. Ingwer Borg and Patrick Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer, New York, 1997.
7. Derek Bridge and Alex Ferguson. Learning weight intervals for classification. In Diarmuid O'Donoghue, editor, *Proc. 12th Irish Conf. on Artif. Intell. & Cognitive Science*, pages 95–104, 2001.

8. Chris Buckley and Gerard Salton. Optimization of relevance feedback weights. In *Proc. 18th Int'l Conf. on Research and Development in Information Retrieval*, pages 351–357, New York, NY, USA, 1995. ACM Press.
9. Stephane Canu and Yves Grandvalet. Adaptive scaling for feature selection in SVMs. In *Adv. in Neural Information Processing Systems 15*, pages 553–560. MIT Press, 2002.
10. T. Cover and P. Hart. Nearest neighbor pattern classification. In *IEEE Trans. on Information Theory*, volume 13, pages 21–27, 1967.
11. Lorcan Coyle and Pádrain Cunningham. Improving recommendation ranking by learning personal feature weights. In *Proc. 7th European Conf. on Adv. in Case-Based Reasoning*, LNAI 3155, pages 560–572, Madrid, Spain, September 2004.
12. Ronan Cummins and Colm O'riordan. Evolving general term-weighting schemes for information retrieval: Tests on larger collections. *Artificial Intelligence Review*, 24(3–4):277–299, 2005.
13. Carlotta Domeniconi and D. Gunopulos. Adaptive nearest neighbor classification using support vector machines. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Adv. in Neural Information Processing Systems 14*, pages 665–672. MIT Press, 2001.
14. Carlotta Domeniconi, Jing Peng, and Dimitrios Gunopulos. An adaptive metric machine for pattern classification. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Adv. in Neural Information Processing Systems 13*, pages 458–464. MIT Press, 2000.
15. Anastasios D. Doulamis and Nikolaos D. Doulamis. A recursive optimal relevance feedback scheme for content based image retrieval. In *Proc. 2001 Int'l Conf. on Image Processing*, volume 2, pages 741–744, 2001.
16. Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley & Sons, New York, 2nd edition, 2001.
17. Christoph F. Eick, Alain Rouhana, Abraham Bagherjeiran, and Ricardo Vilalta. Using clustering to learn distance functions for supervised similarity assessment. *Int'l Scientific Journal of Engineering Applications of Artificial Intelligence*, 19(4):395–401, June 2006.
18. Christoph F. Eick and Nidal M. Zeidat. Using supervised clustering to enhance classifiers. In *Proc. 15th Int'l Symp. on Methodologies for Intelligent Systems*, pages 248–256, Saratoga Springs, NY, May 2005.
19. Amir Globerson and Sam Roweis. Metric learning by collapsing classes. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Adv. in Neural Information Processing Systems 18*, pages 451–458. MIT Press, Cambridge, MA, 2005.
20. Jacob Goldberger, Sam Roweis, Geoffrey Hinton, and Ruslan Salakhutdinov. Neighbourhood components analysis. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Adv. in Neural Information Processing Systems 17*, pages 513–520. MIT Press, Cambridge, MA, 2004.
21. M. Halkidi, D. Gunopulos, N. Kumar, M. Vazirgiannis, and C. Domeniconi. A framework for semi-supervised learning based on subjective and objective clustering criteria. In *Proc. 5th Int'l Conf. on Data Mining*, pages 637–640, 2005.
22. Trevor Hastie and Robert Tibshirani. Discriminant adaptive nearest neighbor classification and regression. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Adv. in Neural Information Processing Systems 8*, pages 409–415. The MIT Press, 1996.

23. Aharon Bar Hillel, Tomer Hertz, Noam Shental, and Daphna Weinshall. Learning distance functions using equivalence relations. In *Proc. 20th Int'l Conf. on Machine Learning*, pages 11–18, 2003.
24. Joshua Zhexue Huang, Michael K. Ng, Honqiang Rong, and Zichen Li. Automated variable weighting in k-means type clustering. *Trans. on Pattern Analysis and Machine Intelligence*, 27(5):657–668, May 2005.
25. Jacek Jarmulak, Susan Craw, and Ray Rowe. Genetic algorithms to optimise CBR retrieval. In *Proc. 5th European Workshop on Adv. in Case-Based Reasoning*, pages 136–147, Tento, Italy, September 2000.
26. Jaz Kandola, John Shawe-Taylor, and Nello Cristianini. Learning semantic similarity. In *Adv. in Neural Information Processing Systems 15*. MIT Press, 2002.
27. Deok-Hwan Kim and Chin-Wan Chung. QCluster: relevance feedback using adaptive clustering for content-based image retrieval. In *Proc. ACM Int'l Conf. on Management of Data*, pages 599–610, New York, NY, USA, 2003. ACM Press.
28. Kenji Kira and Larry A. Rendell. A practical approach to feature selection. In *Proc. 9th Int'l Conf. on Machine Learning*, pages 249–256, Aberdeen, Scotland, UK, 1992. Morgan Kaufmann.
29. Andreas Kohlmaier, Sascha Schmitt, and Ralph Bergmann. A similarity-based approach to attribute selection in user-adaptive sales dialogs. In *Proc. 4th Int'l Conf. on Case-Based Reasoning*, LNAI 2080, pages 306–320, Vancouver, BC, Canada, July 2001.
30. Igor Kononenko. Estimating attributes: Analysis and extensions of RELIEF. In *Proc. 7th European Conf. on Machine Learning*, pages 171–182, 1994.
31. David G. Lowe. Similarity metric learning for a variable-kernel classifier. Technical Report TR-93-43, 1993.
32. Héctor Núñez, Miquel Sànchez-Marrè, and Ulises Cortés. Improving similarity assessment with entropy-based local weighting. In *Proc. 5th Int'l Conf. on Case-Based Reasoning*, LNAI 3689, pages 377–391, Trondheim, Norway, June 2003.
33. Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Hoboken, NJ, 1992.
34. Rong Pan, Qiang Yang, and Lei Li. Case retrieval using nonlinear feature-space transformation. In *Proc. 7th European Conf. on Adv. in Case-Based Reasoning*, LNAI 3155, pages 361–374, Madrid, Spain, September 2004.
35. Dori Peleg and Ron Meir. A feature selection algorithm based on the global minimization of a generalization error bound. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Adv. in Neural Information Processing Systems 17*, pages 1065–1072. MIT Press, Cambridge, MA, 2004.
36. Petra Perner. Why case-based reasoning is attractive for image interpretation. In *Proc. 4th Int'l Conf. on Case-Based Reasoning*, LNAI 2080, pages 27–43, Vancouver, BC, Canada, July 2001.
37. J.J. Rocchio. Relevance feedback in information retrieval. In Gerard Salton, editor, *The SMART Retrieval System–Experiments in Automatic Document Processing*, chapter 14, pages 313–323. Prentice Hall, 1971.
38. Yong Rui, Thomas S. Huang, and Sharad Mehrotra. Relevance feedback techniques in interactive content-based image retrieval. In *Storage and Retrieval for Image and Video Databases*, pages 25–36, San Jose, CA, January 1998.
39. Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, NY, USA, 1983.

40. Sascha Schmitt, Philipp Dopichaj, and Patricia Domínguez-Marín. Entropy-based vs. similarity-influenced: Attribute selection methods for dialogs tested on different electronic commerce domains. In *Proc. 6th European Conf. on Adv. in Case-Based Reasoning*, LNAI 2416, pages 380–394, Aberdeen, Scotland, UK, September 2002.
41. Bernhard Schölkopf. The kernel trick for distances. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Adv. in Neural Information Processing Systems 13*, pages 301–307. MIT Press, 2000.
42. Simon Chi Keung Shiu, Cai Hung Sun, Xi Zhao Wang, and Daniel So Yeung. Maintaining case-based reasoning systems using fuzzy decision trees. In *Proc. 5th European Workshop on Adv. in Case-Based Reasoning*, pages 285–296, Tento, Italy, September 2000.
43. Armin Stahl. Learning feature weights from case order feedback. In *Proc. 4th Int'l Conf. on Case-Based Reasoning*, LNAI 2080, pages 502–516, Vancouver, BC, Canada, July 2001.
44. Armin Stahl and Thomas Gabel. Using evolution programs to learn local similarity measures. In *Proc. 5th Int'l Conf. on Case-Based Reasoning*, LNAI 3689, pages 537–551, Trondheim, Norway, June 2003.
45. Yijun Sun and Jian Li. Iterative RELIEF for feature weighting. In *Proc. 23rd Int'l Conf. on Machine Learning*, pages 913–920, 2006.
46. Eric C.C. Tsang, Simon C.K. Shiu, X.Z. Wang, and Martin Lam. Clustering and classification of cases using learned global feature weights. 2001.
47. Kilian Weinberger, John Blitzer, and Lawrence Saul. Distance metric learning for large margin nearest neighbor classification. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Adv. in Neural Information Processing Systems 18*, pages 1473–1480. MIT Press, Cambridge, MA, 2005.
48. D. Wilson and D. Leake. Maintaining case-based reasoners: Dimensions and directions. *Computational Intelligence*, 17:196–212, 2001.
49. Nirmalie Wiratunga, Ivan Koychev, and Stewart Massie. Feature selection and generalisation for retrieval of textual cases. In *Proc. 7th European Conf. on Adv. in Case-Based Reasoning*, LNAI 3155, pages 806–820, Madrid, Spain, September 2004.
50. Liu Yang, Rong Jin, Rahul Sukthankar, and Yi Liu. An efficient algorithm for local distance metric learning. In *Proc. 21st Nat'l Conference on Artificial Intelligence*, 2006.
51. Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proc. 20th Int'l Conf. on Machine Learning*, pages 856–863, 2003.
52. Zhihua Zhang. Learning metrics via discriminant kernels and multidimensional scaling: Toward expected euclidean representation. In *Proc. 20th Int'l Conf. on Machine Learning*, pages 872–879, 2003.