Using Clustering to Learn Distance Functions For Supervised Similarity Assessment

Christoph F. Eick, Alain Rouhana, Abraham Bagherjeiran, Ricardo Vilalta

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

Abstract Assessing the similarity between objects is a prerequisite for many data mining techniques. This paper introduces a novel approach to learn distance functions that maximizes the clustering of objects belonging to the same class. Objects belonging to a data set are clustered with respect to a given distance function and the local class density information of each cluster is then used by a weight adjustment heuristic to modify the distance function so that the class density is increased in the attribute space. This process of interleaving clustering with distance function modification is repeated until a "good" distance function has been found. We implemented our approach using the k-means clustering algorithm. We evaluated our approach using 7 UCI data sets for a traditional 1-nearest-neighbor (1-NN) classifier and a compressed 1-NN classifier, called NCC, that uses the learnt distance function and cluster centroids instead of all the points of a training set. The experimental results show that attribute weighting leads to statistically significant improvements in prediction accuracy over a traditional 1-NN classifier for 2 of the 7 data sets tested, whereas using NCC significantly improves the accuracy of the 1-NN classifier for 4 of the 7 data sets.

1 Introduction

Many tasks, such as case-based reasoning, cluster analysis and nearest-neighbor classification, depend on assessing the similarity between objects. Defining object similarity measures is a difficult and tedious task, especially in high-dimensional data sets.

Only a few papers center on learning distance function from training examples. Stein and Niggemann [10] use a neural network approach to learn weights of distance functions based on training examples. Another approach, used by [7] and [9], relies on an interactive system architecture in which users are asked to rate a given similarity prediction, and then uses reinforcement learning to enhance the distance function based on the user feedback.

Other approaches rely on an underlying class structure to evaluate distance functions. Han, Karypis and Kumar [4] employ a randomized hill-climbing approach to learn weights of distance functions for classification tasks. In their

approach k-nearest-neighbor queries are used to evaluate distance functions; the k-neighborhood of each object is analyzed to determine to which extend the class labels agree with the class label of each object. Zhihua Zhang [13] advocates the use of kernel functions and multi-dimensional scaling to learn Euclidean metrics. Finally, Hastie et. al. [5] propose algorithms that learn adaptive rectangular neighborhoods (rather than distance functions) to enhance nearest-neighbor classifiers.

There has also been some work that has some similarity to our work under the heading of semi-supervised clustering. The idea of *semi-supervised clustering* is to enhance a clustering algorithm by using side information that usually consists of a "small set" of classified examples. Xian's approach [12] transforms the classified training examples into constraints: points that are known to belong to different classes need to have a distance larger than a given bound. He then derives a modified distance function that minimizes the distance between points in the data set that are known to belong to the same class with respect to these constraints using classical numerical methods ([1] advocates a somewhat similar approach). Klein [6] proposes a shortest path algorithm to modify a Euclidean distance function based on prior knowledge.

This paper introduces an approach that learns distance functions that maximize class density. It is different from the approaches that were discussed above in that it uses clustering and not k-nearest-neighbor queries to evaluate a distance function; moreover, it uses reinforcement learning and not randomized hill climbing or other numerical optimization techniques to find "good" weights of distance functions.

The paper is organized as follows. Section 2 introduces a general framework for similarity assessment. Section 3 introduces a novel approach that learns weights of distance functions using clusters for both distance function evaluation and distance function enhancement. Section 4 describes our approach in more depth. Section 5 discusses results of experiments that analyze the benefits of using our approach for nearest-neighbor classifiers. Finally, Section 6 concludes the paper.

2 Similarity Assessment Framework Employed

In the following a framework for similarity assessment is proposed. It assumes that objects are described by sets of attributes and that the similarity of different attributes is measured independently. The dissimilarity between two objects is measured as a weighted sum of the dissimilarity with respect to their attributes. To be able to do that, a weight and a distance measure has to be provided for each attribute. More formally, define:

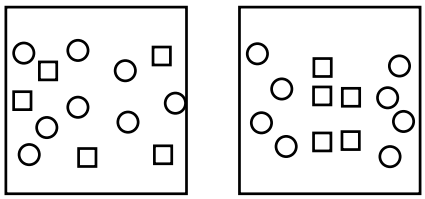
$O = \{o_1, \ldots, o_n\}$	Set of objects whose similarity has to be assessed
$o_{i,j}$	Value of attribute att_j for object $o_i \in O$
Θ_i	Distance function of the i-th attribute
w_i	Weight for the i-th attribute

Based on the definitions in the above table, the distance Θ between two objects o_1 and o_2 is computed as follows:

$$\Theta(o_1, o_2) = \frac{\sum_{i=1}^{m} w_i \Theta_i(o_{1,i}, o_{2,i})}{\sum_{i=1}^{m} w_i}$$

3 Interleaving Clustering and Distance Function Learning

In this section, we will give an overview of our distance function learning approach. Then, in the next section, our approach is described in more detail. The key idea of our approach is to use clustering as a tool to evaluate and enhance distance functions with respect to an underlying class structure. We assume that a set of classified examples is given. Starting from an initial object distance function d_{init}, our goal is to obtain a "better" distance function d_{good} that maximizes class density in the attribute space.



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

Figure 1. Visualization of the Objectives of the Distance Function Learning Process

Fig. 1 illustrates what we are trying to accomplish; it depicts the distances of 13 examples, 5 of which belong to a class that is identified by a square and 8 belong to a different class that is identified by a circle. When using the initial distance function d_{init} we cannot observe too much clustering with respect to the two classes; starting from this distance function we like to obtain a better distance function d_{good} so that the points belonging to the same class are clustered together. In Fig. 1 we can identify 3 clusters with respect to d_{good}, 2 containing circles and one containing squares. Why is it beneficial to find such a distance function d_{good}? Most importantly, using the learnt distance function in conjunction with a k-nearest-neighbor classifier allows us to obtain a classifier with high predictive accuracy. For example, if we use a 3-nearest-neighbor classifier with d_{good} it will have 100% accuracy with respect to leave-one-out cross-validation, whereas several examples are misclassified if d_{init} is used. The second advantage is that looking at d_{good} itself will tell us which features are important for the particular classification problem.

There are two key problems for finding "good" object distance functions:

1. We need an evaluation function that is capable of distinguishing between good distance functions, such as d_{good}, and not so good distance functions, such as d_{init}.
2. We need a search algorithm that is capable of finding good distance functions.

Our approach to address the first problem is to cluster the object set O with respect to the distance function to be evaluated. Then, we associate an error with the result of clustering process that is measured by the percentage of minority examples that occur in the clusters obtained.

Our approach to the second problem is to adjust the weights associated with the i-th attribute relying on a simple reinforcement learning algorithm that employs the following weight adjustment heuristic. Let us assume a cluster contains 6 objects whose distances with respect to att_1 and att_2 are depicted in Fig. 2:

Figure 2. Idea Underlying the Employed Weight Adjustment Approach

If we look at the distribution of the examples with respect to att_1 we see that the average distance between the majority class examples (circles in this case) is significantly smaller than the average distance considering all six examples that belong to the cluster; therefore, it is desirable to increase the weight w_1 of att_1, because we want to drive the square examples "into another cluster" to enhance class purity; for the second attribute att_2 the average distance between circles is larger than the average distance of the six examples belonging to the clusters; therefore, we would decrease the weight w_2 of att_2 in this case. The goal of these weight changes is that the distances between the majority class examples are decreased, whereas distances involving non-majority examples are increased. We will continue this weight adjustment process until we processed all attributes for each cluster; then we would cluster the examples again with the modified distance function (as depicted in Fig. 3), for a fixed number of iterations.

4 Using Clusters for Weight Learning and Distance Function Evaluation

Before we can introduce our weight adjustment algorithm, it is necessary to introduce the notations in Table 1 that are later used when describing our algorithms.

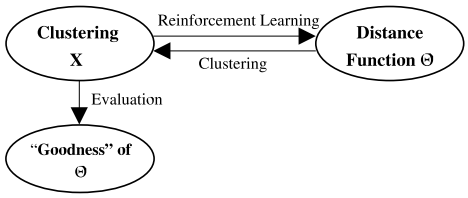


Figure 3. Co-evolving Clusters and Distance Functions

As discussed in [4], searching for good weights of distance functions can be quite expensive. Therefore in lieu of conducting a "blind" search for good weights, we like to use local knowledge, such as density information within particular clusters, to update weights more intelligently. In particular, our proposed approach uses the average distances 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:

w_i	be the current weight of the i-th attribute
σ_i	be the average normalized distances for the examples that belong to the cluster with respect to Θ_i
μ_i	be the average normalized distances for the examples of the cluster that belong to the majority class with respect to Θ_i

Then the weights are adjusted with respect to a particular cluster using formula W:

$$w_i' = w_i + \alpha w_i(\sigma_i - \mu_i) \tag{1}$$

with $1 \geq \alpha > 0$ being the learning rate.

In summary, after a clustering[2] has been obtained with respect to a distance function the weights of the distance function are adjusted using formula 1 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 single examples belonging to different classes.

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

[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.

[2] Clusters are assumed to be disjoint

O	Set of objects (belonging to a data set)
c	Number of different classes in O
$n = \|O\|$	Number of objects in the data set
D_i	Distance matrix with respect to the i-th attribute
D	Object distance matrix for O
$X = \{c_1, \ldots, c_k\}$	Clustering of O with each cluster Ci being a subset of O
$k = \|X\|$	Number of clusters used
$\pi(\Theta, O)$	Clustering algorithm that computes a set of clusters X
$\Psi(\Theta, O) = q(\pi(\Theta, O))$	Evaluation function for Θ using a clustering algorithm π
$q(X)$	Evaluation function that measures the impurity of a clustering

Table 1. Notations used in the description of our algorithms.

distance of 2 to object 4 with respect to Θ_1, and a distance of 3 to object 1 with respect to Θ_3:

$$D_1 \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 3 \\ & 0 & 1 & 2 & 3 & 1 \\ & & 0 & 1 & 2 & 2 \\ & & & 0 & 1 & 3 \\ & & & & 0 & 1 \\ & & & & & 0 \end{bmatrix} \quad D_2 \begin{bmatrix} 0 & 1 & 1 & 1 & 5 & 1 \\ & 0 & 1 & 1 & 5 & 1 \\ & & 0 & 1 & 5 & 5 \\ & & & 0 & 5 & 1 \\ & & & & 0 & 5 \\ & & & & & 0 \end{bmatrix} \quad D_3 \begin{bmatrix} 0 & 3 & 3 & 2 & 2 & 3 \\ & 0 & 3 & 2 & 2 & 3 \\ & & 0 & 2 & 2 & 2 \\ & & & 0 & 1 & 3 \\ & & & & 0 & 1 \\ & & & & & 0 \end{bmatrix} \quad D \begin{bmatrix} 0 & 1.67 & 2 & 2 & 3.67 & 2.33 \\ & 0 & 1.67 & 1.67 & 3.33 & 1.67 \\ & & 0 & 1.33 & 3 & 3 \\ & & & 0 & 2.33 & 2.33 \\ & & & & 0 & 2.33 \\ & & & & & 0 \end{bmatrix}$$

The object distance matrix D is next computed using:

$$D = \frac{w_1 D_1 + w_2 D_2 + w_3 D_3}{w_1 + w_2 + w_3}$$

First, the average cluster and average inter-majority object distances for each attribute have to be computed; we obtain: $\sigma_1 = 2$, $\mu_1 = 1.3$; $\sigma_2 = 2.6$, $\mu_2 = 1$; $\sigma_3 = 2.2$, $\mu_3 = 3$. The average distance and the average majority examples distance within the cluster with respect to Θ are: $\sigma = 2.29$, $\mu = 1.78$. Assuming $\alpha = 0.2$, we obtain the new weights: $\{1.14{*}0.33333, 1.32{*}0.33333, 0.84{*}0.33333\}$. After the weights have been adjusted for the cluster, the following new object distance matrix D is obtained:

$$D = \begin{bmatrix} 0 & 1.51 & 1.86 & 1.95 & 3.89 & 2.20 \\ & 0 & 1.51 & 1.60 & 3.55 & 1.51 \\ & & 0 & 1.26 & 3.20 & 3.20 \\ & & & 0 & 2.60 & 2.20 \\ & & & & 0 & 2.60 \\ & & & & & 0 \end{bmatrix}$$

After the weights have been adjusted for the cluster, the average inter-object distances have changed to: $\sigma = 2.31$, $\mu = 1.63$. As we can see, the examples

belonging to the majority class have moved closer to each other (the average majority class example distance dropped by 0.15 from 1.78), whereas the average distances of all examples belonging to the cluster increased very slightly, which implies that the distances involving non-majority examples (involving objects 4, 5 and 6 in this case) have increased, as intended.

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

$$w_i' = w_i + \alpha\lambda w_i(\sigma_i - \mu_i)$$

with λ being defined as the number of minority examples in the cluster over the average number of minority examples per clusters.

For example, if we had 3 clusters that contain examples belonging to 3 different classes with the following class distributions (9, 3, 0), (9, 4, 4), (7, 0, 4); the average number of minority examples per cluster in this case is (3+8+4)/3=5; therefore, λ would be 3/5=0.6 when adjusting the weights of the first cluster, 8/5 when adjusting the weights of the second cluster, and 4/5 when adjusting the weights in the third cluster.

As explained earlier, our approach searches for good weights using a weight adjustment heuristic that was explained in the previous section; however, how do we know which of the found distance functions is the best? In our approach a distance function is evaluated based on the impurity of the clustering X obtained with respect to the distance function to be evaluated. As explained earlier, impurity is measured by the percentage of minority examples that occur in the different clusters of solution X. A minority example is an example that belongs to a class different from the most frequent class in its cluster.

5 Experimental Evaluation

5.1 Data Sets Used and Preprocessing

We tested the distance function learning approach for a benchmark consisting of the following 7 datasets: DIABETES, VEHICLE, HEART-STATLOG, GLASS, HEART-C, HEART-H, and IONOSPHERE that were taken from the University of Irvine's Machine Learning Benchmark [2]. Table 2 gives a short summary for

Dataset	n	Classes	Attributes
DIABETES	768	2	8
VEHICLE	846	4	18
HEART-STATLOG	270	2	13
GLASS	214	6	9
HEART-C	303	5	13
HEART-H	294	5	13
IONOSPHERE	351	2	34

Table 2. Data Sets Used in the Experimental Evaluation

each data set. All seven data sets only contain numerical attributes. The numerical attributes in each data set were normalized by using a linear interpolation function that assigns 1 to the maximum value and 0 to the minimum value for that attribute in the data set.

5.2 Algorithms Evaluated in the Experiments

In the experiments conducted, we compared the performance of two different 1-nearest-neighbor classifiers that use the learnt weights with a traditional 1-nearest-neighbor classifier that considers all attributes to be of equal importance. Moreover, we also compare the results with a decision tree classifier. Details about the algorithm evaluated will be given in this section.

Our distance function learning approach does not only learn a distance function Θ, but also obtains a centroid and a majority class for each cluster. These (centroid, majority class) pairs can be used to construct a 1-nearest-neighbor classifier that we call *nearest centroid classifier* (NCC) in the following. NCC is based on the idea that a cluster's centroid is used as the representative for a cluster. NCC classifies new examples by assigning the majority class of the closest centroid to it. NCC uses the learnt distance function Θ to determine the closest centroid. A nearest centroid classifier can be viewed as a "compressed" 1-nearest-neighbor classifier that operates on a set of $k < n$ cluster representatives, rather than using all training examples.

In particular, in the experiments the following four algorithms were tested for the 7 data sets that have been described in the previous section:

1-NN 1-nearest-neighbor classifier that uses all examples of the training set and does not use any attribute weighting.

LW1NN 1-nearest-neighbor classifier with attribute weighting (same as 1-NN but weights are learnt using the methods we described in Sections 3 and 4)

NCC 1-nearest-neighbor classifier that uses k (centroid, majority class) pairs, instead of all objects in the data set; it also uses attribute weighting.

C4.5 uses the C4.5 decision tree learning algorithm that is run with its default parameter settings.

5.3 Experimental Results

Experiments were conducted using the WEKA toolkit [11]. The accuracy of the four algorithms was determined by running 10-fold cross validation 10 times. Table 2 shows the accuracy results averaged over the ten runs of cross validation for each data set/classification algorithm pair.

The weight learning algorithm was run for 200 iterations and best weight combination found with respect to q was reported. We used 1/j (were j is the number of attributes) as the initial weights; that is, attributes are assumed to have the "same" importance initially. Moreover, after each iteration weights were normalized so that the sum of all weights always adds up to 1. The learning rate α was linearly decreased from 0.6 at iteration 1 to 0.3 at iteration 200.

A supervised clustering algorithm [3] was used to determine the k-values for the DIABETES, and VEHICLE data sets, and for the other data sets k-values were set to 5 times the number of classes in the data set. The decision tree and 1-NN classifiers used in the experiments are the standard classifiers that accompany the WEKA toolkit. The remaining algorithms use two modified WEKA algorithms: the k-means clustering and 1-NN algorithms. The modifications to each permit the use of attribute weights when computing object similarity.

We chose the 1-NN classifier as the reference algorithm for the experiments and indicated statistically significant improvements[3] of other algorithms over 1-NN in bold face in Table 2. The table also indicates the number of objects n in each data set, as well as the parameter k that was used when running k-means. If we compare the 1-nearest-neighbor classifier with our attribute weighting approach (LW1NN), we see that the weight learning approach demonstrates significant improvements of more than 3.5% in accuracy for the GLASS, and IONOSPHERE data sets (also outperforming C4.5 for those data sets), but does not show any statistically significant improvements for the other five data sets.

Dataset	n	k	1-NN	LW1NN	NCC	C4.5
DIABETES	768	35	70.62	68.89	**73.07**	74.49
VEHICLE	846	64	69.59	69.86	65.94	72.28
HEART-STATLOG	270	10	76.15	77.52	**81.07**	78.15
GLASS	214	30	69.95	**73.5**	66.41	67.71
HEART-C	303	25	76.06	76.39	**78.77**	76.94
HEART-H	294	25	78.33	77.55	**81.54**	80.22
IONOSPHERE	351	10	87.1	**91.73**	86.73	89.74

Table 3. Accuracy for the 4 Classification Algorithms

[3] Statistical significance was determined by a paired t-test on the accuracy for each of the 10 runs of 10-fold cross validation.

Using NCC, on the other hand, demonstrates significant improvements in accuracy for the DIABETES, HEART-STATLOG, HEART-C, and HEART-H data sets, which is quite surprising considering the small number of representatives used by the compressed classifier. For example, for the HEART-C data set the 303*0.9 objects[4] in a training set were replaced by 25 (centroid, majority class)-pairs and the distance function that considers every attribute of equal importance was replaced with the learnt distance function. Moreover, the accuracies of the three HEART data sets are at least 1.5% higher than those reported for C4.5. Again, not surprisingly, for the other data sets reducing the number of objects used by a nearest-neighbor classifier, results in a reduced accuracy, sometimes significantly less.

Each run of the weight learning process for each fold was captured through a set of graphs. Fig. 4 depicts one of those run summaries for the DIABETES data set. It shows how cluster impurity and weights changed during that particular run. The DIABETES data set has eight attributes and therefore eight weights Weight0,...,Weight7 that are initially set to 0.125. The initial impurity was about 25% at the beginning of the run and the minimum impurity of 21% was reached approximately at iteration 180; the other graphs depict how the 8 weights of the 8 attributes changed iteration by iteration. For example, Weight3 dropped from its initial value of 0.125 to approximately 0.03 at the end of the run, whereas Weight5 increased from 0.125 to approximately 0.38 near the end of the run. As mentioned earlier, weights are normalized prior to clustering; therefore, the sum of the 8 weights always adds up to 1.

6 Conclusion

The paper presented a novel approach to learn distance functions with respect to an underlying class structure that uses clusters for both distance function evaluation and distance function enhancement. We also proposed a novel weight adjustment heuristic that adapts weights of distance functions based on class density information in clusters.

The proposed weight learning approach was evaluated using a benchmark consisting of 7 data sets. Our empirical results suggest that our attribute weighting approach enhanced the prediction accuracy of a 1-nearest-neighbor (1-NN) classifier significantly for 2 of the 7 data sets. However, our approach does not only learn a distance function, but also provides a centroid and a majority class for each cluster. These (centroid, majority class) pairs can be used to construct a nearest centroid classifier (NCC) that is a "compressed" nearest-neighbor classifier. Surprisingly, although it uses only a small percentage of the examples belonging to a data set using NCC lead to significant improvements in accuracy for 4 of the 7 data sets.

We claim that our approach facilitates the tailoring of distance functions supporting different perspectives with respect to the data set to be analyzed. For

[4] 10-fold cross validation only uses 90% of the examples of a data set for training.

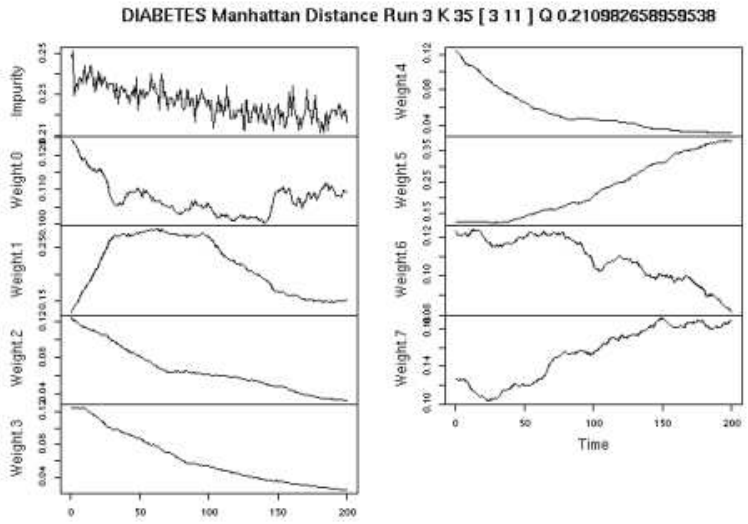


Figure 4. Display of a Run of the Weight Learning Algorithm for the VEHICLE Data Set

example, one user might be interested in analyzing products with respect to the time spend for their delivery whereas another analyst is interested in analyzing products with respect to the profit they produced. Using our approach we would generate a distance function for each analysis perspective that, we believe, is particularly important for data warehousing applications, such as those that rely on OLAP-technology.

References

1. Bar-Hillel, A., Hertz, T., Shental., N. & Weinshall, D. (2003). Learning Distance Functions Using Equivalence Relations, in Proc. ICML'03, Washington D.C.
2. Blake, C.L. & Merz, C.J. (1998). UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]Irvine, CA: University of California, Department of Information and Computer Science.
3. Eick, C., Zeidat, N. (2005). *Using Supervised Clustering to Enhance Classifiers*, in Proc. 15th International Symposium on Methodologies for Intelligent Systems (ISMIS), Saratoga Springs, New York.
4. Han, E.H., Karypis, G. & Kumar, V. (1999). Text Categorization Using Weight Adjusted nearest-neighbor Classification, Lecture Notes in Computer Science.
5. Hastie, T. & Tibshirani, R. (1996). Disriminant Adaptive Nearest-Neighbor Classification, in IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 18, pp. 607-616.

6. Klein, D., Kamvar, S.-D. & Manning, C. (2002). From instance-level Constraints to Space-level Constraints: Making the Most of Prior Knowledge in Data Clustering, in Proc. ICML'02, Sydney, Australia.
7. Kira K. & Rendell L. (1992). A practical Approach to Feature Selection, in Proc. 9th Int. Conf on Machine Learning.
8. MacQueen, J (1967). Some methods for classification and analysis of multi-variate observations, in Proc. 5th Berkeley Symposium Math., Stat., Prob., 1:281-297.
9. Salzberg, S. (1991). A nearest Hyperrectangle Learning Method, Machine Learning.
10. Stein, B. & Niggemann, O. (2001). Generation of Similarity Measures from Different Sources, in Proc. Fourteenth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems Springer Verlag.
11. Witten, I. & Eibe, F. (2000). *Data Mining: Practical machine learning tools with Java implementations*, by Ian H. Witten and Eibe Frank, Morgan Kaufmann, San Francisco.
12. Xing, E.P., Ng, A., Jordan, M. & Russell S. (2003). Distance Metric Learning with Applications to Clustering with Side Information. Advances in Neural Information Processing 15, MIT Press.
13. Zhang, Z. (2003). Learning Metrics via Discriminant Kernels and Multi-Dimensional Scaling: Toward Expected Euclidean Representation, in Proc. ICML'03, Washington D.C.