A TREE-LIKE REPRESENTATION OF THE TRAINING SET FOR CLASSIFICATION TIME REDUCTION

J. Lobato, J.S. Sánchez, J.M. Sotoca
Dept. Llenguatges i Sistemes Informàtics, Universitat Jaume I
Av. Sos Baynat s/n, E-12071 Castelló
Spain

ABSTRACT

In this paper, we introduce a new algorithm to efficiently search for neighbors of a test sample. The main advantage of this technique over other search algorithms refers to the fact that the one proposed here can be applied to any kind of neighborhood. Thus the algorithm results in a tree-like data structure adapted to the specific type of neighborhood to be further employed by a distance-based classifier. Experiments over several synthetic data sets illustrate the benefits of using the method introduced in this paper in terms of classification time.

KEY WORDS

Pattern Recognition; Classification; Neighborhood; Tree Design Algorithm; Space Partition.

1. INTRODUCTION

Non-parametric classification by means of a distance measure is one of the earliest methods used in Pattern Recognition. The Nearest Neighbor (NN) rule [1] is an appropriate example of this kind of classifiers. Given a set of *N* previously labeled prototypes (namely, training set) in a *d*-dimensional feature space, this rule assigns to a given sample the same class than the closest prototype in the set. More generally, the *k*-NN rule maps any sample to the pattern class most frequently represented among the *k* closest neighbors.

In the recent years, many other examples of distance-based classification have been proposed. In this context, the *k*-NCN (Nearest Centroid Neighbors) rule along with classifiers based on two cases of proximity graphs [2], the Gabriel Graph (GG) and the Relative Neighborhood Graph (RNG), constitute three appropriate alternatives to the classical *k*-NN scheme. It is worth pointing out that in general, these approaches achieve better classification results than the NN rules.

Despite of the simplicity and effectiveness of distance-based classifiers, it is well known that they suffer from some practical drawbacks, such as needing a lot of memory and computational resources for large training sets. This effect becomes even more important in the case of the *k*-NCN and proximity graph-based approaches. Paradoxically, while numerous investigations have been carried out on *k*-NN techniques in order to find the nearest neighbor of an unknown test sample with as few computations as possible, nothing has been made in the direction of efficient *k*-NCN and proximity graph-based classifiers.

For example, related to the *k*-NN scheme, several strategies have been proposed to devise fast algorithms to search for the nearest neighbor [3-7]. Other alternatives focus on the use of some data structures which allow a more efficient search than computing distances from a given test sample to all prototypes in the training set. Most of these approaches are based on a certain partition of the *d*-dimensional feature space. In particular, *kd*-tree methods [7,8] are a popular tool to obtain a tree-like classifier from a decomposition of the feature space with hyperplanes parallel to the axes. A similar solution [9,10] is concerned to an implementation of the NN rule in the form of a neural network classifier, from the Voronoi Diagram associated with the training set.

The aim of the algorithm introduced in this paper is in the line of building some data structure that allows a more efficient classification of test samples. Nevertheless, as far as we know, the schemes previously proposed have concentrated on only *k*-NN problems, that is, the algorithm makes use of some geometrical properties to obtain a data structure that can be further employed to efficiently classify test samples by means of the *k*-NN rule. Conversely, the algorithm here proposed can be applied to any kind of neighborhood, resulting in a tree-like data structure. Thus, for a given distance-based classifier, the corresponding tree is built from the training set and allows focusing the search for neighbors of a test sample among a very reduced number of prototypes.

2. NEIGHBORHOOD REALIZATIONS

As already mentioned in the previous section, apart from the *k*-NN classification rule, many other distance-based classifiers have been proposed in the literature. They differ from each other in the particular neighborhood definition used for classification. In this context, those that define neighborhood by using not only proximity but also symmetry are specially remarkable. Such approaches try to look for neighbors close enough (in the basic distance sense) but also homogeneously or symmetrically distributed around the input point. For this reason, they are referred to as surrounding neighborhood.

Among the surrounding neighborhood realizations, one encounters the NCN concept [11] and those derived from two examples of proximity graphs, the GG and the RNG [12]. The general idea consists of taking into account not only the distance from a given test sample to its neighbors, but also their geometrical distribution around it. It is apparent that this allows obtaining a more reliable information before making a decision about the class membership of a test sample.

The NCN concept is based on the general idea that the neighborhood of a point should be subject to two constraints. First, by the proximity criterion, the k neighbors of a sample p must be as near as possible. And second, by the symmetry criterion, their centroid must be also as close to p as possible.

Let p be an input point whose k-NCN should be found in a set of d-dimensional points $X = \{x_1, ..., x_n\}$. In practice, these NCN can be algorithmically obtained as follows [11]:

1. The first NCN of p is its NN, q_1.
2. The i-th neighbor, q_i, $i \leq 2$, is such that the centroid of this and all previously selected neighbors, $q_1,...,q_{i-1}$ is the closest to p.

On the other hand, in the case of the neighborhood obtained from the GG and the RNG, two samples x and y are said to be graph neighbors in a given proximity graph, $G = (V,E)$, if there exists an edge $(x, y) \in E$ between them. Taking into account the definitions of GG and RNG [12], the graph neighborhood of a given point requires that no other point lies inside the union of the zones of influence (i.e., hypersphere or lune of influence) corresponding to all its graph neighbors. From this neighborhood relation, it seems possible to completely surround a prototype by means of a variable number of neighbors (that is, all its graph neighbors).

3. A TREE-LIKE REPRESENTATION

In this section, we introduce a new technique to represent a given training set in the form of a particular tree that divides the whole feature space into a number of regions, each one containing a reduced subset of training prototypes. The resulting tree-like structure [13] is hereafter called *Tree of Influence on Regions* (TIR).

It is to be noted that the tree design algorithm needs as input a classification rule (along with any tuning parameter as the neighborhood size in k-NN and k-NCN schemes) and a distance function. In other words, the algorithm to build a TIR can be employed with any kind of neighborhood.

Before describing the tree design algorithm and the classification procedure by using the resulting tree, we will define some preliminary concepts.

Definition 1. A training prototype p has not influence on a given region Z if no sample in Z is in the neighborhood of p. Otherwise, it is said that p is a prototype of influence on Z.

Definition 2. In each dimension j, a prototype p has a lower bound, say L, and an upper bound, say R, that define its *interval of influence* in such a dimension, say $int_j(p)$.

Definition 3. The *projection segment* of a feature j is a list with the lower and upper bounds of the intervals of influence corresponding to all training prototypes in dimension j, that is, $pseg_j = \cup\ int_j(p),\ \forall\ p \in X$. Each projection segment has $2n$ elements, being n the number of training prototypes.

3.1 The TIR Design Algorithm

In this section, we present the algorithm to build TIR and discuss some properties of this tree. Let $X = \{x_1, ..., x_n\}$ be a training set, where $x_i = \{x_{i1}, ..., x_{id}\}$ is a given prototype in a d-dimensional space. We employ a classification rule *alg* and a distance function *dist*. Moreover, we should define any additional tuning parameter of the classification scheme *alg* (e.g., the neighborhood size k in k-NN and k-NCN classifiers).

Through a normalization of the training set X, the coordinates of points x_i are transformed into normalized points $x_i^N = \{x_{i1}^N, ..., x_{id}^N\}$ by means of the following expression:

$$x_{ij}^N = \frac{x_{ij} - min(x_j)}{max(x_j) - min(x_j)} \qquad \textbf{(1)}$$

where $max(x_j)$ and $min(x_j)$ denote the maximum and the minimum values in the training set X for the dimension j.

After, we are to obtain *the intervals of influence* of the prototypes $x_i^N \in X^N$ (i.e., the normalized training set) in each dimension. This is the most critical step and in fact, the temporal cost of the algorithm is mainly due the cost of performing it. To this end, it is defined a grid of samples uniformly distributed (with an arbitrary number of m samples in each dimension). Thus, this grid will have a total of m^d elements.

So, for each sample in the grid, we have to perform the following steps:

1. For each sample in the grid, obtain its neighbors using the classifier rule *alg*, and the distance criterion *dist*.
2. For each neighbor, update its intervals of influence making that the region bounded by such intervals reach to the sample of the grid used in this iteration.

Basically, the TIR design algorithm can be written as follows:

1. Define the intervals of influence for each prototype in each dimension.
2. Define the *d* projection segments and sort these according to the position of the projected points. This suppose to load a structure [*L, B, R*] where *B* refers to the points between *L* and *R*.
3. Let *S* be the projection of the prototype in the projection segment. For each dimension, move the point until finding the optimal dimension and position. The optimum is the one that minimizes the expression $\text{abs}(S/2 - L) + \text{abs}(S/2 - R)$.
4. If the optimum position $> S/2$ or $L < 2$ or $R < 2$, the process stops because this node corresponds to a leaf. All prototypes associated with such a leaf. Otherwise, the prototypes represented by *B* are associated with this node.
5. In order to optimize the TIR, since there are prototypes in the upper nodes that have not influence on all regions represented by their descendent nodes, we expand a prototype in an upper node to their two children if it has no influence on all descendent nodes. This process is performed for all prototypes at each node, beginning from the root to a leaf of the TIR.

3.2 Classifying with TIR

From the design algorithm introduced in the previous section, it seems clear enough that classification by means of TIR will be more efficient than directly using the training set (that is, in the form of a list). In fact, as already explained, TIR allows focusing on a very reduced number of training prototypes when searching for neighbors of a given test sample in its classification process.

In brief, classifying a test sample $y = \{y_1, ..., y_d\}$ will basically consist of searching for its neighbors among the (few) prototypes at each node belonging to a path from the root to a leaf. Let *cnode* be a given node, let d_{cnode} be its split dimension and p_{cnode} its split position, let $List_{cnode}$ denote the list of prototypes associated with *cnode*, and let $Left_{cnode}$ and $Right_{cnode}$ be the left and right children, respectively. Then, the classification process will be performed as follows:

```
1. Normalize a test sample $y$ using the equation 1, obtain the normalize values $y^N = \{y_1^N,...,y_d^N\}$.
2. Let $X_{\text{sub}} = \phi$ and $cnode$ = root.
3. Let $X_{\text{sub}} = X_{\text{sub}} \cup List_{\text{cnode}}$.
4. Let $i = d_{\text{cnode}}$.
   a. If $cnode$ is a leaf, then go to Step 5.
   b. If $y^N_i \le p_{\text{cnode}}$ then $cnode = Left_{\text{cnode}}$ and go to Step 3.
   c. If $y^N_i > p_{\text{cnode}}$ then $cnode = Right_{\text{cnode}}$ and go to Step 3.
5. Classify $y$ using the prototypes in $X_{\text{sub}}$.
```

4. EXPERIMENTS AND RESULTS

Several experiments have been performed to properly assess the merits and possible drawbacks of the technique just described. Here the main goal is to show that the new representation of the training set allows reducing the classification time of several distance-based classifiers.

Such a behavior is studied using a well-known synthetic database with high overlap [14]. All classification schemes were implemented in *Java* programming language using the same code to make them as similar as possible. The experiments were performed on a 2400-MHz Intel Pentium IV.

The synthetic database consists of a collection of seven data sets that correspond to the same problem but with dimensionality ranging from 2 to 8. All the features are designed to be equally effective in terms of their discrimination potential thus making the analysis a function of only the size of the subset used and not of its specific features. The samples are divided into two classes representing multivariate normal distributions with zero mean and standard deviation 1 and 2 in all dimensions, respectively. Each class contains a total of 2,500 samples. Each original database has randomly been divided into 2,500 training samples and 2,500 test samples.

Experiments consist of classifying the test set by directly using the prototypes in the training set (this first option is here referred to as *classical algorithm*) and by means of the TIR that has been built with the technique proposed in Section 3. The neighborhood-based classifiers used in this paper has been 3-NN, 3-NCN, GG and RNG, while the distance function has been the Euclidean distance in all cases.

Apart from the classification time required by each option, we are also interested in comparing their classification accuracies because, in its present form, TIR results in an approximation to those neighborhood-based classifiers.

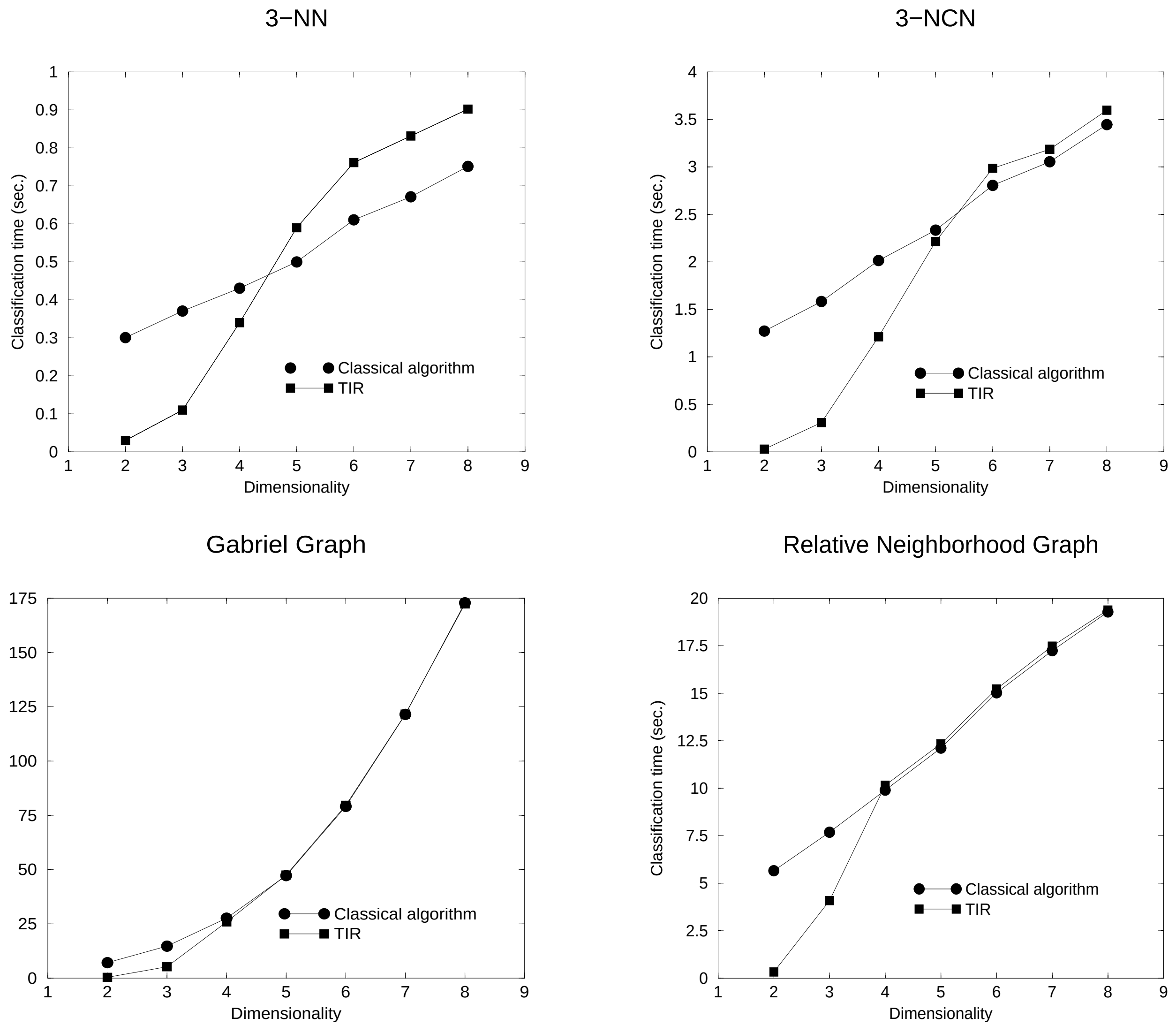

Figure 1. Classification time with varying dimensionality for several neighborhoods.

Figure 1 illustrates the time required for classifying the artificial test sets by using 3-NN, 3-NCN, GG and RNG in two different situations. First, when the training set is directly employed by the classifier, that is, the so-called classical algorithm. An second, when the training set has been preprocessed by the scheme presented in Section 3, and the resulting tree structure has been further employed to search for neighbors of the test samples during classification.

As can be seen, the algorithm proposed here considerably reduces the classification time in all cases (that is, with all neighborhood realizations). Nevertheless, it is to be noted that, in its present form, such an algorithm should be specially applied to the lowest dimensions, in the sense that time savings are clearly obtained up to dimension 4 because of the way in which the intervals of influence are now defined.

Table 1 reports the classification accuracy given by 3-NN, 3-NCN, GG and RNG approaches at each dimension, both when the training set has been directly employed for classifying the test set and when the training prototypes have been preprocessed by the TIR design algorithm.

Table 1. Classification accuracy.

		2D	3D	4D	5D	6D	7D	8D
3-NN	Class.	67.0	72.7	76.6	80.2	80.2	81.7	80.4
	TIR	66.9	72.7	77.2	79.8	80.6	81.6	80.7
3-NCN	Class.	66.8	72.5	77.1	80.6	82.5	85.2	86.4
	TIR	67.1	72.3	77.2	80.7	82.9	84.7	86.6
GG	Class.	67.9	75.6	81.5	84.3	86.2	88.4	89.4
	TIR	68.3	75.4	81.3	84.2	86.2	88.4	89.4
RNG	Class.	65.4	72.2	77.4	81.3	82.8	84.4	86.2
	TIR	65.5	72.2	77.2	81.8	82.9	84.6	86.1

For each neighborhood realization, one can observe that differences between both approaches are not statistically significant and therefore, it seems that TIR becomes a good approximation to neighborhood search. In fact, the possible differences between them are only due to the fact that, in the case of using the tree design algorithm, the search space is obtained from the training set and consequently, some test samples could fall out of such a search space. It is expected that future research in this direction can result in an even more accurate approximation.

5. CONCLUDING REMARKS

In this paper, a new technique for efficiently finding neighbors has been proposed. The algorithm allows designing a tree-like data structure that will be further employed for the classification of test samples. One of the most important properties of such a method refers to the fact that it can be applied to any kind of neighborhood. In particular, the NCN and two examples of graph neighborhood have been used in this paper.

The experiments here carried out have shown the significant reduction in classification time achieved when preprocessing the training set by means of the tree design algorithm. In its present form, the method proposed here obtains the highest time reduction rate in the lowest dimensions. Consequently, our current work is primarily addressed to improve several steps of the algorithm so that it can achieve the same considerable benefits as in the lowest dimensions.

On the other hand, the current TIR corresponds to an approximation to any kind of neighborhood and therefore, there exist some differences in classification accuracy with respect to the exact neighbors search. Nevertheless, experimental results have shown that such differences are not statistically significant.

6. ACKNOWLEDGEMENTS

This work has partially been funded by the Spanish CICYT (Department of Science and Technology) under grant TIC2000-1703-C03-03 and by the Fundació Caixa Castelló-Bancaixa under grant P1-1B2002-07.

REFERENCES

[1] T.M. Cover and P.E. Hart, Nearest neighbour pattern classification, *IEEE Trans. on Information Theory, 13*(1), 1967, 21-27.

[2] J.S. Sánchez, F. Pla and F.J. Ferri, On the use of neighbourhood-based non-parametric classifiers, *Pattern Recognition Letters, 18*(11-13), 1997, 1179-1186.

[3] K. Fukunaga and P.M. Narendra, A branch and bound algorithm for computing *k*-nearest neighbors, *IEEE Trans. on Computers, 24*(7), 1975, 750-753.

[4] E. Vidal, An algorithm for finding nearest neighbours in (approximately) constant average time, *Pattern Recognition Letters, 4*(3), 1986, 145-157.

[5] S.O. Belkasim, M. Shridhar and M. Ahmadi, Pattern classification using an efficient KNNR, *Pattern Recognition, 25*(10), 1992, 1269-1274.

[6] A. Djouadi and E. Bouktache, A fast algorithm for the nearest-neighbor classifier, *IEEE Trans. on Pattern Analysis and Machine Intelligence, 19*(3), 1997, 277-282.

[7] P.J. Grother, G.T. Candela and J.L. Blue, Fast implementations of nearest neighbour classifiers, *Pattern Recognition, 30*(3), 1997, 459-465.

[8] J.H. Friedman, J.L. Bentley and R.A. Finkel, An algorithm for finding best matches in logarithmic expected time, *ACM Trans. on Mathematical Software, 3*(2), 1977, 209-226.

[9] O.J. Murphy, Nearest neighbor pattern classification perceptrons, *Proc. IEEE, 78*(10), 1990, 1595-1598.

[10] N.K. Bose and A.K. Garga, Neural network design using Voronoi diagrams, *IEEE Trans. on Neural Networks, 4*(5), 1993, 778-787.

[11] B.B. Chaudhuri, A new definition of neighborhood of a point in multi-dimensional space, *Pattern Recognition Letters, 17*(1), 11-17, 1996.

[12] J.W. Jaromczyk and G.T. Toussaint, Relative neighborhood graphs and their relatives, *Proc. IEEE, 80*(9), 1502-1517, 1992.

[13] L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone, *Classification and regression trees* (New York: Chapman & Hall, 1984).

[14] P.M. Murphy and D.W. Aha, UCI repository of machine learning databases, Dept. of Information and Computer Science, University of California, Irvine, CA, 1991.