From the Nearest Neighbour Rule to Decision Trees[†]

J.S. Sánchez[1], F. Pla[1] and F.J. Ferri[2]

[1] *Dept. d'Informàtica. Universitat Jaume I.*
Campus Penyeta Roja, E-12071 Castelló. Spain.
{sanchez,pla}@uji.es, +34 64 345676

[2] *Dept. d'Informàtica i Electrònica. Universitat de València.*
Dr. Moliner 50, E-46100 Burjassot (València). Spain.
ferri@uv.es, +34 6 3864768

Abstract. This paper proposes an algorithm to design a tree-like classifier whose result is equivalent to that achieved by the classical Nearest Neighbour rule. The procedure consists of a particular decomposition of a *d*-dimensional feature space into a set of convex regions with prototypes from just one class. Some experimental results over synthetic and real databases are provided in order to illustrate the applicability of the method.

Key words. Nearest Neighbour; Decision Tree; Algorithm.

1 Introduction

Non-parametric classification by means of a distance measure is one of the earliest methods used in Pattern Recognition. The *Nearest Neighbour* (NN) rule [1] is an appropriate example of this kind of classifiers. Given a set of N previously labelled 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.

Unfortunately, although the NN rule is simple and powerful, its implementation can become computationally expensive in terms of storage space and computing time, requiring the computation of N distances for one test sample. Consequently, for a large set of prototypes of high dimensionality, the use of this technique becomes exceedingly unrealistic. In order to overcome this practical drawback, a reasonable aim for a classification system consists of using an algorithm able of finding the nearest neighbour of an unknown sample with as few computations as possible.

In fact, a number of investigations have been concerned with minimising the computational burden of the NN classifiers. On the one hand, several strategies have

[†] This work was partially supported by grants P1B96-13 (Fundació Caixa Castelló-Bancaixa), AGF95-0712-C03-01 and TIC95-676-C02-01 (Spanish CICYT).

been proposed to devise efficient algorithms to search for the nearest neighbour [2], [3], [4] On the other hand, efforts have been directed to select a suitable reduced set of prototypes, usually referred to as *condensed set* [5], that leads to approximately the same performance than the NN rule using the whole training set [6], [7], [8].

The purpose of this paper is to introduce an alternative approach for the application of the NN rule, without computing distances from a given sample to prototypes in the training set. That is, instead of trying to improve the direct computation of the nearest neighbour, we propose an alternative classification scheme whose result will be identical to that of the conventional NN rule. In order to achieve this goal, we provide a deterministic algorithm to design a particular decision tree from partitioning of the feature space into convex regions, using the hyperplanes of the Voronoi boundaries associated with a training set.

2 Decision Trees

Decision or *classification trees* (DT) partition a d-dimensional feature space into regions, basically hyperrectangles parallel to the axes. Among these, the most important are the binary DTs [9], since they have just two children per node and are thus easiest to manipulate and update. We recall the simple terminology of books on data structures. The top of a binary tree is called the root. Each node has either no child (in that case, it is called terminal node or leaf), a left child, a right child, or a left child and a right child. Each node is the root of a tree itself. The trees rooted at the children of a node are called the left and right subtrees of that node. The depth of a node is defined as the length of the path from the node to the root. The height of a tree is the maximal depth of any node.

In a binary DT, each node represents a set in the feature space. Moreover, each node has exactly two or zero children. So, if a node u represents the set A and its children u', u'' represent A' and A'', respectively, then we require that $A = A' \cup A''$ and $A' \cap A'' = \varnothing$. The root represents the entire feature space, and the leaves, taken together, form a partition of it.

Each leaf has a class label assigned in some way, and each non-terminal node (or decision node) represents an intermediate or partial decision rule, defined by a linear function. During the classification process, each test sample is dropped into the tree at the root node. The sample follows a path through the tree according to the decision nodes and, it is thus assigned to the class of the leaf reached.

Classification by means of a binary DT has got some attractive advantages. First, it establishes a decision technique that is easy to analyse and understand. Second, it uses a systematic way for calculating the class to assign to an unknown sample. On the other hand, DT is much faster in terms of computing time than the traditional NN decision rule, what obviously constitutes an important benefit from a practical point of view. Moreover, there exist some prominent architectures on which tree classifiers can be mapped to facilitate hardware implementation, like on FPGA (Field Programmable Gate Arrays) boards.

Note that the tree structure is usually data dependent, as well, and indeed, it is in the construction itself where methods differ. For instance, the operation proposed here is based on a tree decomposition of a d-dimensional feature space into *homogeneous convex regions* (that is, regions which contain prototypes from just one class), by means of the hyperplanes that define the Voronoi boundaries associated with a set of prototypes.

3 The DT Design Approach

The design algorithm iteratively searches for convex regions derived from combinations with a finite number of the hyperplanes which define the Voronoi boundaries related to a set of prototypes (that is, those hyperplanes which separate Voronoi regions with different class labels). Thus, each of those convex regions will be assigned to the class of its nearest neighbour among the set of prototypes. This process is straightforward and gives rise to a binary tree structured classifier equivalent to the NN rule in terms of classification result. So, for this reason, the DT obtained from the design algorithm is hereafter called *Nearest Neighbourhood Tree* (NNT).

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of n prototypes belonging to k distinct classes in a certain d-dimensional feature space and, let $\nabla = \{H_1, H_2, \ldots, H_m\}$ be the set of m hyperplanes which define the Voronoi boundaries associated with the set X.

The algorithm starts by considering the entire d-dimensional feature space as a convex region, $\Re_1$. Taking into account that any hyperplane H_i can divide the feature space into two half-spaces (namely, the *positive* half-space H_i^+ and the *negative* one H_i^-), if we now take the first hyperplane, $H_1 \in \nabla$, then $\Re_1$ will be divided into two partial convex regions. In this case, H_1 is the decision rule (linear function) associated with the root node of the current tree configuration.

Further, the algorithm goes on taking hyperplanes from the set ∇. At each stage, the split criterion will consist of testing whether the given hyperplane divides some of those partial convex regions $\Re_j$, that is, those regions defined by the half-spaces derived from the decision rules (or hyperplanes) corresponding to the nodes in each path in the current NNT, or not. When a hyperplane H_i results in a new partition over any of those partial regions $\Re_j$, the current NNT will be expanded with a new non-terminal node, whose decision rule will be the hyperplane H_i just tested, i.e., the convex region $\Re_j$ is divided into two new smaller convex regions. This process is iterated by using all the hyperplanes of the Voronoi boundaries $H_i \in \nabla$ (the stop-splitting rule).

The problem just introduced is solved by using the *Simplex* method for Linear Programming. In order to test whether a hyperplane $H_i \in \nabla$ can divide a certain convex region or not, we formulate a linear program which consists of optimising an arbitrary linear objective function subject to a set of constraints. These constraints will be the inequations (a set of constraints of the form $\geq$ or $\leq$) corresponding to the half-spaces which define the given convex region, and the equation (a constraint of

the form =) of that hyperplane H_i. The hyperplane H_i does not divide a convex region if and only if the corresponding linear program is infeasible.

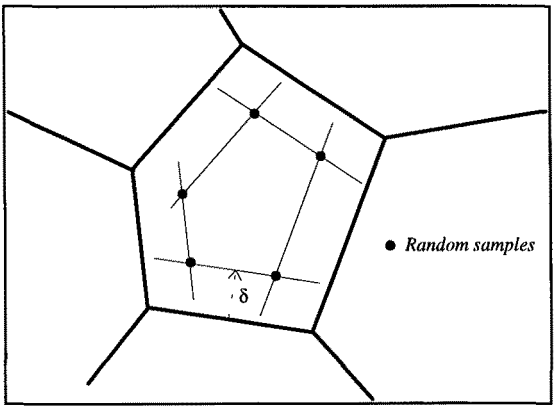

Fig. 1. Generation of samples within a convex region

With respect to the procedure for assigning a class label to leaves in the NNT, we can generate a sample within each convex region (see Fig. 1), and searching for its nearest neighbour among the prototypes in the input set: the class label of its nearest neighbour is assigned to the corresponding convex region. In order to generate a sample belonging to a given convex region, we can move inside an arbitrary distance, δ, the edges of such a region and then, take any vertex of the resulting region. This vertex will be the optimal feasible vector for a new linear program with an arbitrary objective function subject to a set of constraints (the inequations of the half-spaces which define the inside convex region).

Finally, the size of the resulting tree can be reduced by means of a simple pruning approach. That is, if both children nodes (leaves) of a non-terminal node have assigned the same class label, it means that the hyperplane associated with such a decision node is separating two convex regions which are from the same class and therefore, it is indeed irrelevant for the discrimination process. Hence, such a non-terminal node can be assigned to the class attached to its children nodes and, these two can be deleted from the initial tree structure.

3.1 Algorithm

The method just described can be summarised in the following algorithm:

Step 1: Let $\Re$ denote the set of convex regions in the tree structure after each iteration. Begin with the complete d-dimensional feature space as the unique region in $\Re$.

Step 2: *Splitting*. For each $H_i \in \nabla$ do

Step 2.1: For each $\Re_j \in \Re$ do

Step 2.1.1: If $\Re_j$ can be linearly separated by H_i:

- Designate H_i as decision function for the non-terminal node represented by $\Re_j$.
- Make $\Re_{j+1} = \Re_j \cap H_i^-$ and $\Re_{j+2} = \Re_j \cap H_i^+$ the convex regions that represent the children nodes of $\Re_j$.
- Delete $\Re_j$ from $\Re$ and put $\Re_{j+1}$, $\Re_{j+2}$ in $\Re$.

Step 3: *Assignment*. For each final convex region $\Re_j \in \Re$ do

Step 3.1: Assign to $\Re_j$ a class label.

Step 4: *Pruning*. Prune the resulting tree.

With respect to the computational burden, it is worth mentioning that the cost of the algorithm is of order 2^m, where m denotes the number of hyperplanes which support the Voronoi boundaries, thus the cost does not depend either the dimensionality of the feature space or directly on the number of samples in the training set. Nevertheless, if this set is large, the Voronoi boundaries can become more detailed, and therefore the number of those hyperplanes related to the Voronoi boundaries increases.

4 Experimental Results

Some examples about how to design a NNT from a set of prototypes are here provided. These experiments are all in a 2-dimensional feature space in order to make easier their graphical representation. However, it should be pointed out that the method proposed in Section 3 does not take advantage of any particular 2-dimensional property, that is, the design algorithm can be applied to any d-dimensional problem.

Experiment 1. For the first example, it is available a training set of three prototypes, $X = \{(0,4),(2,2),(4,4)\}$, corresponding to two distinct classes in the plane. Therefore, let $S_1 = \{(0,4),(2,2)\}$ be the subset of points that belong to class 1, and let $S_2 = \{(4,4)\}$ be the subset of points in class 2. Here, the hyperplanes that define the Voronoi boundaries are H_2 and H_3, hence the set of the hyperplanes required to design the proposed NNT classifier will be $\nabla = \{H_2, H_3\}$. The Voronoi diagram of these three points in the plane has been represented in Fig. 2.

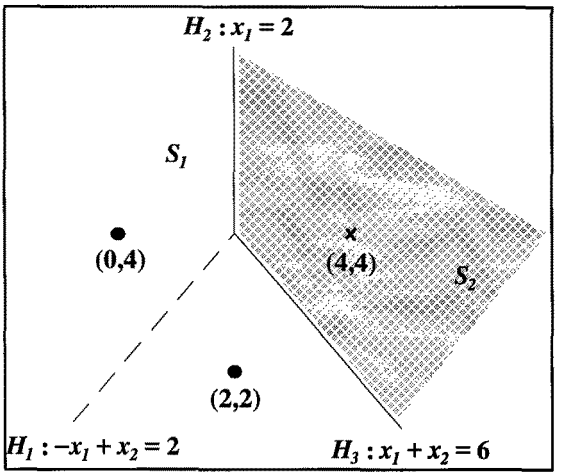

Fig. 2. The Voronoi diagram for three points in the plane

The first hyperplane in ∇, H_2, is assigned to the root node. Obviously, H_2 divides the space into two half-spaces say, $\Re_1 = H_2^-$ and $\Re_2 = H_2^+$, and therefore produces two children nodes of the root node. In order to determine whether the next hyperplane H_3 divides $\Re_1$ and $\Re_2$, we must formulate two different linear programs with an arbitrary objective function, $x_1 + x_2 = 0$, subject to a set of constraints. The constraints for the first program (that is, the linear program for the convex region $\Re_1$) are

$$x_1 \leq 2$$
$$x_1 + x_2 = 6$$

which correspond to the half-space H_2^- and the hyperplane H_3, respectively. Since there exists a feasible vector which satisfies the given constraints, it means that the hyperplane H_3 divides the region defined by the half-space H_2^- into two new half-spaces. Therefore, H_3 is designated as decision rule for the left child node of the root node.

Analogously, the Simplex method is now applied to the other convex region $\Re_2$ in order to test whether the hyperplane H_3 can divide the convex region $\Re_2$. In this case, the constraints are

$$x_1 \geq 2$$
$$x_1 + x_2 = 6$$

corresponding to the half-space H_2^+ and the hyperplane H_3, respectively. In this case, there also exists a feasible vector that optimises the objective function, hence H_3 is designated as decision rule for the right child node of H_2.

Since there are no more hyperplanes in the set of Voronoi boundaries ∇, the resulting convex regions are

$$\Re_1 = (H_2^- \cap H_3^-)$$
$$\Re_2 = (H_2^- \cap H_3^+)$$
$$\Re_3 = (H_2^+ \cap H_3^-)$$
$$\Re_4 = (H_2^+ \cap H_3^+)$$

Now, a sample for each convex region $\Re_i$ is generated. This can be performed by applying the *Simplex* method once again, but moving the boundaries of those convex regions towards their centres, as follows

$$\Re_1^d = (H_2^- - \delta \cap H_3^- - \delta)$$
$$\Re_2^d = (H_2^- - \delta \cap H_3^+ + \delta)$$
$$\Re_3^d = (H_2^+ + \delta \cap H_3^- - \delta)$$
$$\Re_4^d = (H_2^+ + \delta \cap H_3^+ + \delta)$$

Obviously, the solution vector after applying the Simplex algorithm to this shrunk convex region corresponds to a point that belongs to the original convex region. Further, the algorithm searches for the nearest neighbour of the solution vector for convex regions $\Re_i^d$, and assigns to each of these convex regions the same class label than its nearest neighbour among the set of prototypes X. Thus, $\Re_1^d$, $\Re_2^d$ and $\Re_4^d$ are assigned to S_1, and $\Re_3^d$ is assigned to S_2. This results in the tree structure represented in Fig. 3.

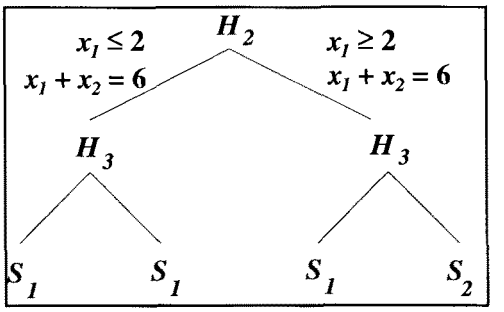

Fig. 3. The non-pruned NNT

Finally, the size of this NNT can be reduced by means of the pruning stage: the left branch of the root node can be pruned because both leaves are from the class S_1. Therefore, the left child node of the root node is transformed into a terminal node with the same class label, S_1, as its children nodes. The resulting pruned tree has been drawn in Fig. 4.

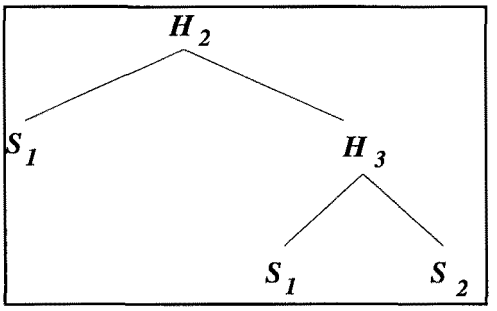

Fig. 4. The final NNT

Experiment 2. The second experiment is carried out on a real database, which consists of values from RGB images with a resolution of 256 x 256 pixels. This database was previously used in order to study colour segmentation to locate oranges in outdoor scenes under daylight conditions. There are 6,386 samples with two different attributes (coordinates φ and θ of the colour vectors in the RGB space) and three classes (oranges, sky and leaves). A combined editing-condensing technique was previously applied to the original set of prototypes, retaining only 61 of those samples. The Voronoi diagram corresponding to this reduced set contains a total of 172 hyperplanes in two dimensions (see Fig. 5). From all these hyperplanes, only 65 belong to the Voronoi boundaries.

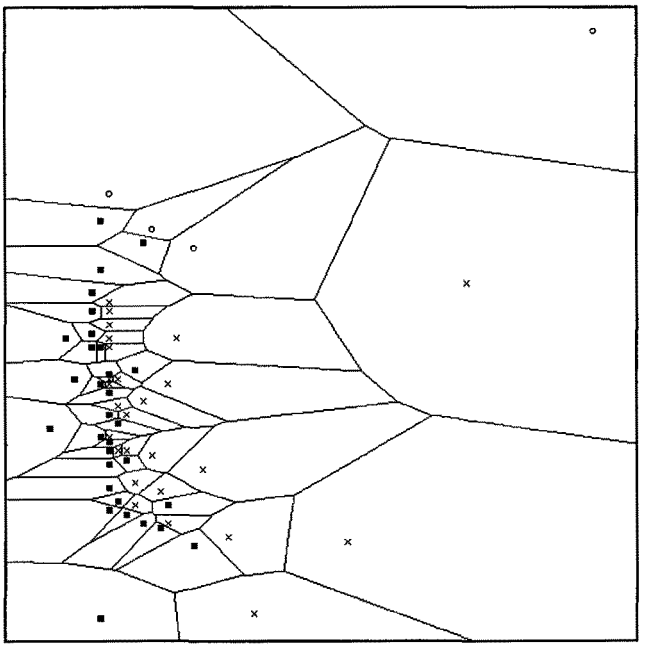

Fig. 5. The Voronoi diagram for points in Experiment 2

Applying the design algorithm, this results in a non-pruned binary tree with 2,262 leaves. After pruning, the corresponding NNT contains a total number of 510 leaves. In this case, the tree structure has not been drawn because of difficulty to represent such a number of nodes.

5 Conclusions and Extensions

A technique for the design of a specific kind of binary DT, whose classification result is identical to that achieved by the classical NN rule, has been introduced. In this paper, the tree-like NN classifier proposed has been called *Nearest Neighbourhood Tree*, trying to reflect both its structure and the kind of information used for classification. The design procedure is based on the use of the set of hyperplanes which define the Voronoi boundaries derived from the Voronoi diagram of the prototypes and, consists of an iterative process which decomposes a d-dimensional feature space into a number of convex regions, each one finally assigned to a certain class. The decision rules for non-terminal nodes in the tree correspond to those hyperplanes of the Voronoi boundaries, while leaves are the class labels associated with each one of the resulting convex regions.

The algorithm provides a systematic way to obtain a DT structure with exactly the same classification result as the well-known NN decision rule, but avoiding the traditional approach to compute distances from a test sample to prototypes in the training set. The cost of using the proposed NNT is expected to be lower since it only requires to compare a point with a hyperplane at most m times. Moreover, note that we have the possibility to speed up the process using the properties of binary DTs to be implemented in parallel architectures or easily programmable hardware devices, like FPGA-based boards.

Currently, the ordering of the set of hyperplanes (those which correspond to the Voronoi boundaries) constitutes a crucial step. In fact, different orderings can produce radically different trees after pruning. Nevertheless, finding an optimal solution (that is, an optimal ordering to build smaller trees) could be an intractable problem. Some experiments with respect to the ordering problem can be found in [11].

As an extension to this work, another interesting application consists of using the design procedure as a learning technique for induction of oblique DTs [10], which constitutes one of the most critical steps for the construction of tree classifiers [9]. The usual technique to build a DT from a training set, Ψ, consists of dividing this set into two subsets, Ψ_1 and Ψ_2. The first one is used to construct the complete tree structure, while the second one is utilised to select an appropriate tree from the series of trees resulting from pruning the complete tree.

The advantages of applying our algorithm to induce oblique DTs are basically concerned to the fact that the resulting DT would be more accurate because training is carried out by using all prototypes in the input data set. Comparative studies with performance of other oblique tree training methods is left to further work.

References

1. Cover, T.M. and Hart, P.E.: Nearest neighbour pattern classification, *IEEE Trans. on Information Theory* **13** (1967) 21-27.
2. Fukunaga, K. and Narendra, P.M.: A branch and bound algorithm for computing *k*-nearest neighbors, *IEEE Trans. on Computers* **24** (1975) 750-753.
3. Vidal, E.: An algorithm for finding nearest neighbours in (approximately) constant average time, *Pattern Recognition Letters* **4** (1986) 145-157.
4. Djouadi, A. and Bouktache, E.: A fast algorithm for the nearest-neighbor classifier, *IEEE Trans. on Pattern Analysis and Machine Intelligence* **19** (1997) 277-282.
5. Devijver, P.A. and Kittler, J.: *Pattern Recognition: A statistical approach.* Prentice Hall, Englewood Cliffs, NJ (1982).
6. Hart, P.E.: The condensed nearest neighbor rule, *IEEE Trans. on Information Theory* **14** (1968) 515-516.
7. Toussaint, G.T., Bhattacharya, B.K. and Poulsen, R.S.: The application of Voronoi diagrams to nonparametric decision rules, In: *Proc. of Computer Science & Statistics: 16th Symposium of the Interface*, Atlanta, GA (1984) 97-108.
8. Chen, C.H. and Józwik, A.: A sample set condensation algorithm for the class sensitive artificial neural network, *Pattern Recognition Letters* **17** (1996) 819-823.
9. Breiman, L., Friedman, J.H., Olshen, R.A. and. Stone, C.J.: *Classification and regression trees.* Chapman & Hall, New York, NY (1984).
10. Heath, D., Kasif, S. and Salzberg, S.: Learning oblique decision trees, In: *Proc. of the 13th International Joint Conference on Artificial Intelligence* (1993) 1202-1207.
11. Sánchez, J.S.: *Aprendizaje y clasificación basados en criterios de vecindad. Métodos alternativos y análisis comparativo.* Ph.D. Thesis. Dept. of Computer Science, University Jaume I, Castelló, Spain (1998).