Approximate Nearest Neighbour Search with the Fukunaga and Narendra Algorithm and Its Application to Chromosome Classification

Francisco Moreno-Seco, Luisa Micó, and Jose Oncina*

Dept. Lenguajes y Sistemas Informáticos
Universidad de Alicante, E-03071 Alicante, Spain,
{paco,mico,oncina}@dlsi.ua.es

Abstract. The nearest neighbour (NN) rule is widely used in pattern recognition tasks due to its simplicity and its good behaviour. Many fast NN search algorithms have been developed during last years. However, in some classification tasks an exact NN search is too slow, and a way to quicken the search is required. To face these tasks it is possible to use approximate NN search, which usually increases error rates but highly reduces search time.
In this work we propose using approximate NN search with an algorithm suitable for general metric spaces, the Fukunaga and Narendra algorithm, and its application to chromosome recognition. Also, to compensate the increasing in error rates that approximate search produces, we propose to use a recently proposed framework to classify using k neighbours that are not always the k nearest neighbours. This framework improves NN classification rates without extra time cost.

Keywords: Approximate Nearest Neighbour, Pattern Recognition, Chromosome Recognition.

1 Introduction

The nearest neighbour (NN) rule classifies an unknown sample into the class of its nearest neighbour according to some similarity measure (a *distance*). Despite its simplicity, classification accuracy is usually enough for many tasks. However, some tasks may require finding the k nearest neighbours in order to improve classification rates, thus the NN rule has been generalized to the k-NN rule [3]. Many classification tasks represent data as vectors and use one of the Minkowsky metrics as the distance, usually the L_2 (Euclidean distance). However, there are other tasks where a vector representation is not suitable, and thus other distance measures are used: string distance, tree distance, etc.

Although heavily used in pattern recognition, the NN rules have been also of interest for other fields such as data mining and information retrieval, which usually involves searching in very large databases and facing with high dimensional

* The authors wish to thank the Spanish CICyT for partial support of this work through project TIC2000–1703-CO3-02.

A. Sanfeliu and J. Ruiz-Shulcloper (Eds.): CIARP 2003, LNCS 2905, pp. 322–328, 2003.

data. Whenever the classification task requires large training sets or expensive distance measures, the simple exhaustive search for the NN becomes unpractical. To overcome some of these problems, a large number of fast NN search algorithms [5,4,13,11,2,10] have been developed; most of them have been easily extended to find the k-NN. However, the requirement of finding exactly the k-NN involves higher computing effort (dependent on the value of k).

For some tasks finding exactly the NN (even using a fast NN search algorithm) may become too slow; some approximate NN search algorithms [1] have been proposed to face these tasks, yielding slightly worse classification rates but obtaining much lower classification times.

Recently [9], a framework for approximate k-NN classification based on approximation-elimination fast NN search algorithms has been proposed. The main idea in that work is to modify a NN search algorithm keeping a sorted array with the prototypes whose distance to the sample has been computed during the search (the *selected* prototypes), and classify the sample by voting among the nearest k prototypes found while searching for the NN, including the NN itself. Those prototypes are called the k nearest selected prototypes (k-NSN).

In this work we have applied the ideas from [1] to the Fukunaga and Narendra algorithm, which has been implemented using a priority queue to allow approximate search. Then, to improve classification rates we propose to use either the k-NSN classification scheme or the k-NN scheme; while the first improves classification rates without increasing classification times, the latter obtains better classification rates than k-NSN but at an extra time cost that depends on the value of k.

2 The Fukunaga and Narendra Algorithm Implemented Using a Priority Queue

The algorithm from Fukunaga and Narendra [5] is a classic NN search algorithm. In the preprocessing phase, a tree is built from the training set, using some hierarchical clustering algorithm. In [5] the k-means algorithm is suggested for clustering the set at each level in the tree, and due to this suggestion the Fukunaga and Narendra algorithm is often considered suitable only for Euclidean spaces. However, if a more general clustering algorithm is used instead, the search algorithm is suitable for any metric space.

In the tree, each non-leaf node p contains a representative M_p of a set of prototypes S_p, a radius R_p (the maximum of the distances between M_p and all the other prototypes in S_p), and l children. Leaf nodes contain only a representative M_p and the set of prototypes S_p. The search phase traverses the tree using a branch and bound scheme. At each node, the distances from the representatives of its children to the sample are computed and stored. Given a child p, the pruning condition is:

$$d_{nn} + R_p < d(x, M_p) \tag{1}$$

where x is the sample and d_{nn} is the distance to the nearest neighbour found so far. For all non-pruned nodes, the search continues, starting with the nearest child. When the node p is a leaf, all the prototypes stored in the node are tested: if they can not be the nearest neighbour, they are pruned; otherwise, its distance to the sample is computed and the nearest neighbour is updated if necessary. Given a leaf node p, the pruning condition for a prototype $x_i \in S_p$ is:

$$d_{nn} + d(x_i, M_p) < d(x, M_p) \tag{2}$$

Please note that $d(x, M_p)$ has been previously computed, and $d(x_i, M_p)$ is computed and stored during the building of the tree, so this condition does not involve new distance computations.

The original formulation of the Fukunaga and Narendra algorithm [5] is usually reformulated in a more intuitive recursive way, but in this work we have implemented it using a priority queue that allows for approximate search: after computing all the distances to the children, all non-pruned nodes are stored in a priority queue (similar to the one used in [1]), using $d(x, M_p) - R_p$ as the key for the queue (see equation 1). Then, the closest element from the queue is extracted and compared (again) with d_{nn}; if the current node key is greater than d_{nn}, the search is finished as all the nodes in the queue are farther from the sample than the current nearest neighbour (see figure 1 for details).

The Fukunaga and Narendra algorithm can be extended to find exactly the k-NN with a couple of simple modifications: first, let d_{nn} be the distance to the kth NN instead of the distance to the NN. Second, each time a distance is computed, store it in a sorted array of the k-NN distances (if possible). As the value of d_{nn} in the pruning condition changes, the time expended by the algorithm to find exactly the k-NN increases in a quantity that depends on the value of k.

3 Approximate Search and Classification

The condition labelled as (a) in the figure 1 is the condition to finish the search: if the nearest (to the sample) element in the queue has a key m that is greater than the current distance to the nearest neighbour d_{nn}, then the nodes in the queue (including the one who has just been extracted) can not contain the nearest neighbour and the search may be finished.

Applying a technique similar to that in the work by S. Arya and D.M. Mount [1], the condition (a) in figure 1 may be transformed into:

$$\textbf{if} \quad (1+\epsilon)m > d_{nn} \quad \textbf{or} \ldots \tag{3}$$

This new condition (with $\epsilon > 0$, obviously) allows to finish the search when the current nearest neighbour is not too far from the nearest neighbour. Using this new condition, the search will become faster, but the classification rate will become slightly worse. As it may be expected, the faster the search, the worse the classification rate will be, thus the choice of the value for ϵ should be a trade-off between classification time and accuracy.

```
function pqsearch
    input    t (tree)
             x (unknown sample)
    output   nn ∈ P (x's nearest neighbour in P)
begin
     insertPQ(t,0)                          //insert the root of the tree in the queue
     endsearch := false ; B := ∞
     while not endsearch do
         (t, m) := extractMinPQ()           //extract node t with minimum key m
(a)      if m > d_nn or emptyPQ() then
             endsearch := true
         else
             for all p = Child(t) do
                 let M_p be the representative of p, and R_p the radius of p
                 d_p := d(x, M_p)
                 if d_p < d_nn then                    // Updating nearest neighbour
                     d_nn := d_p ; nn := p
                 endif
                 if d_p ≤ d_nn + R_p then                          // non-pruned child
                     if Leaf(p) then
                         for all prototype x_i ∈ S_p do
                             if d_p ≤ d(x_i, M_p) + d_nn then
                                 d_{x_i} := d(x, x_i)
                                 if d_{x_i} < d_nn then    // Updating nearest neighbour
                                     d_nn := d_{x_i} ; nn := x_i
                                 endif
                             endif
                         endfor
                     else
                         insertPQ(p,d_p − R_p)
                     endif
                 endif
             endfor
         endif
     endwhile
end pqsearch
```

Fig. 1. Fukunaga and Narendra algorithm using a priority queue

On the other hand, classification rates may be improved using more than just the nearest neighbour found in the search. If we use the k-NN, the search will become slower, so we need a way to improve classification without increasing classification time. In [9] it is showed that storing the closest k prototypes whose distance to the sample is computed during a (non-approximate) NN search (the k nearest *selected* neighbours, the k-NSN), and classifying the sample by voting among these prototypes improves significantly classification rates, yielding rates

similar to those of a k-NN classifier with the classification time of a NN classifier (finding exactly the k-NN requires an extra overhead).

In this work we present some preliminary results of the application to the Fukunaga and Narendra algorithm of a combination of the two ideas above: approximate search using ϵ to improve speed, and approximate k-NN classification (that is, k-NSN classification) in order to improve classification rates (approximate NN search usually produces higher error rates). Two main changes have been made to the Fukunaga and Narendra's algorithm: the use of a priority queue in the search to allow approximate NN search, and storing the k nearest prototypes visited during the search (the so called k-NSN), in order to classify the sample by voting among them.

4 Experiments

We have developed a set of experiments with a chromosome database [8,7,6] that contains 4400 samples coded as strings. We have chosen to use the Levenshtein distance [12] to measure the distance between two chromosomes in this task. The database has been divided into two sets of 2200 samples each, and two experiments have been performed using one of them for training and the other one for test. The tree has been chosen to be a binary tree containing only one prototype at each leaf, and the k-medians algorithm has been used to recursively partition the training set to build the tree.

The experiments were repeated for several values of ϵ and, in order to test the effect of using more than just one neighbour to classify, the k-NSN and k-NN schemes were used for classification; the values of k ranged from 1 to 15. Figure 2 shows the evolution of both error rate and classification time of a 1-NN search for increasing values of ϵ (1-NSN and 1-NN results are the same by definition). The results for $k = 15$ are plotted in figure 3, which shows as a reference the 1-NN error rate and classification time.

As the figures 2 and 3 show, the choice of a value for ϵ depends on the amount of allowable error increase, or on the amount of speed increase required. Also, using more than just one neighbour to classify improves error rates, thus allowing a higher value for ϵ. If classification time is critical for the task, then the best choice seems to be the k-NSN, which requires no extra time over a $k = 1$ search and improves NN classification rates. However, using k-NN produces lower error rates but with a certain time overhead. For the classification task presented in this work, the overhead is low due mainly to the low value of k; higher values of k have been tested but did not yield better classification results.

5 Conclusions and Future Work

We have combined two techniques to speed up the classification time and to improve classification rates, and we have tested that combination on a classic and widely known fast NN search algorithm, the Fukunaga and Narendra algorithm. The results show that the classification process using approximate search

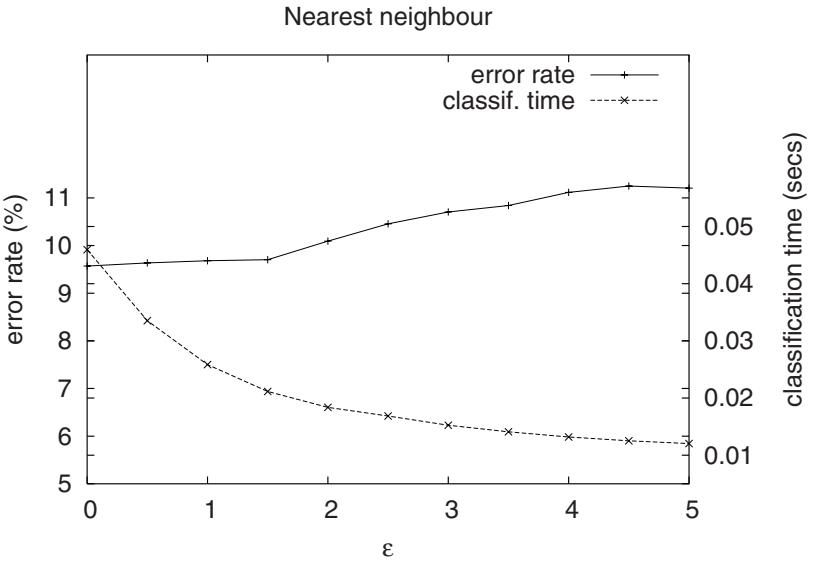

Fig. 2. Error rates and classification times for several values of ϵ, for a 1-NN search.

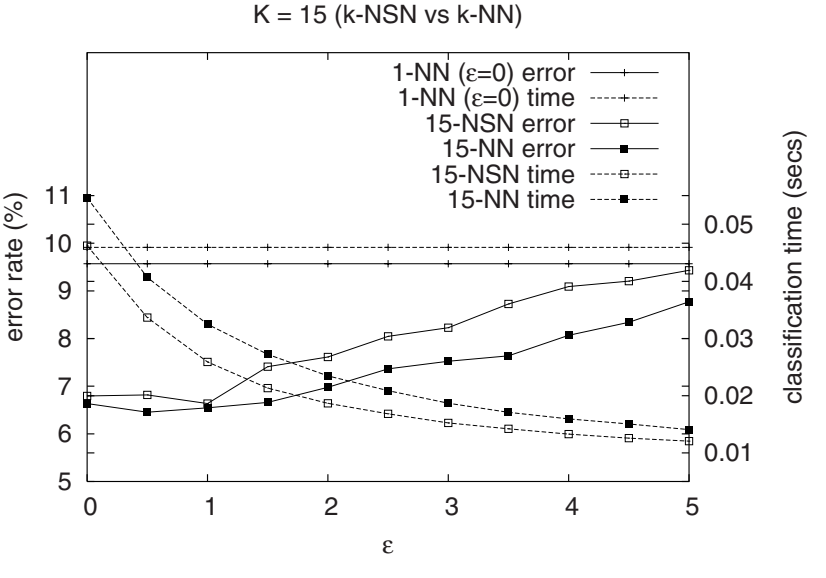

Fig. 3. Comparison of error rates and classification times for several values of ϵ, for $k = 15$.

(with $\epsilon > 0$) is considerably faster, about four times faster for the chromosomes database. Also, the classification rates obtained may be improved using either k-NSN or k-NN classification schemes, which yield to rates always better than those of a non-approximate NN classifier, even with high values for ϵ.

As for the future we plan to apply the same techniques to tree-based NN search algorithms other than Fukunaga and Narendra's. We will also study the relation between the value of ϵ and the classification time and accuracy, using also other databases, either synthetic or real.

Acknowledgments. The authors wish to thank Alfons Juan for providing us with the chromosomes database.

References

1. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.: An optimal algorithm for approximate nearest neighbor searching. Journal of the ACM (1998) **45** 891–923
2. Brin, S.: Near Neighbor Search in Large Metric Spaces. Proceedings of the 21^{st} VLDB Conference (1995) 574–584
3. Duda, R., Hart, P.: Pattern Classification and Scene Analysis. Wiley (1973)
4. Friedman, J.H., Bentley, J.L., Finkel, R.A.: An algorithm for finding best matches in logarithmic expected time. ACM Transactions on Mathematical Software (1977) **3** 209–226
5. Fukunaga, K., Narendra, M.: A branch and bound algorithm for computing k–nearest neighbors. IEEE Trans. Computing (1975) **24** 750–753
6. Granum, E., Thomason, M.G.: Automatically inferred Markov network models for classification of chromosomal band pattern structures. Cytometry (1990) **11** 26–39
7. Granum, E., Thomason, M.G., Gregor, J.: On the use of automatically inferred Markov networks for chromosome analysis. In Automation of Cytogenetics, C. Lundsteen and J. Piper, eds., Springer-Verlag (1989) 233–251
8. Lundsteen, C., Phillip, J., Granum, E.: Quantitative analysis of 6985 digitized trypsin G-banded human metaphase chromosomes. Clinical Genetics (1980) **18** 355–370
9. Moreno-Seco, F., Micó, L., Oncina, J.: Extending fast nearest neighbour search algorithms for approximate k-NN classification. Pattern Recognition and Image Analysis. Lecture Notes in Computer Science, F.J. Perales et al (Eds.) vol. 2652, Springer-Verlag (2003) 589–597
10. Nene, S., Nayar, S.: A Simple Algorithm for Nearest Neighbor Search in High Dimensions. IEEE Transactions on Pattern Analysis and Machine Intelligence (1997) **19**(9) 989–1003
11. Vidal, E.: New formulation and improvements of the Nearest-Neighbour Approximating and Eliminating Search Algorithm (AESA). Pattern Recognition Letters (1994) **15** 1–7
12. Wagner, R.A., Fischer, M.J.: The String-to-String Correction Problem. Journal of the Association for Computing Machinery (1974) **21**(1) 168–173
13. Yianilos, P.N.: Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. ACM-SIAM Symposium on Discrete Algorithms (1993) 311–321