

Some improvements on NN based classifiers in metric spaces

Francisco Moreno-Seco, Luisa Micó, 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) and k -nearest neighbour (k -NN) classification rules have been widely used in Pattern Recognition due to its simplicity and good behaviour. Exhaustive nearest neighbour search may become unpractical when facing large training sets, high dimensional data or expensive dissimilarity measures (distances). During the last years a lot of fast NN search algorithms have been developed to overcome those problems, and many of them are based on traversing a data structure (usually a tree) testing several candidates until the nearest neighbour is found.

When these algorithms are extended to find the k nearest neighbours, the classification time increases with the value of k . In this paper we propose a new classification rule that makes use of the prototypes that are selected by these algorithms in a 1-NN search as candidates to nearest neighbour. To illustrate the behaviour of this rule, several fast and widely known NN search algorithms have been extended with it, obtaining classification results similar to those of a k -NN ($k > 1$) classifier without the extra computational overhead. Also, previous work on approximate NN search for vector spaces has been extended to algorithms suitable for general metric spaces, and has been combined with the new classification rule.

Keywords Nearest Neighbour, Classification Rule, Approximate Search

1 Introduction

Given a set P of prototypes, where each $p \in P$ belongs to one of a finite set of classes C , the nearest neighbour (NN) rule classifies an unknown sample into the class of its nearest neighbour in P according to some similarity measure (a *distance*). Despite its simplicity, the classification accuracy is usually enough for many Pattern Recognition tasks. However, some tasks may require lower classification error rates, and usually the k -NN rule [1] is used as a generalisation of the NN rule. The k -NN classification rule is also simple: find the k nearest neighbours of the sample and

```

 $d_{nn} := \infty$ 
do until the training set  $P$  is empty
   $p_i := \operatorname{argmin}_{p \in P} \operatorname{Aprox}(x, p)$  // Approximation
(a)  $d := d(x, p)$ 
  if  $d < d_{nn}$  then
     $nn := p ; d_{nn} := d$  // Update the current NN
(b)  $P := P - \{ q : q \notin E(x, d_{nn}) \}$  // Elimination
  endif
enddo

```

Figure 1: Approximation and elimination framework.

classify it by voting with the classes of the k nearest neighbours, i.e., assign the majority class to the sample.

Although initially used in Pattern Recognition, the NN rule has 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 dimensionality data. Whenever the classification task requires large training sets, expensive distance measures or high dimensionality, the simple exhaustive search for the NN becomes unpractical. To overcome some of these problems, a large number of fast NN search algorithms [2, 3, 4, 5, 6, 7] have been developed. Many of these algorithms are suitable for any kind of prototype representation (vectors, strings, trees, ...) which allows to define a distance that holds the properties of a metric, that is, they do not assume that the prototype is a vector and thus they do not make use of the coordinates (see the work by Chávez et al. [8] for a review on NN search algorithms in metric spaces).

Most of these fast NN search algorithms may be easily extended to find the k -NN. However, the requirement of finding exactly the k -NN involves higher computing effort, and that effort increases with the value of k .

Several search algorithms can fit into an approximation and elimination framework [9], that can be formulated as in figure 1. The search process is seen as an iterative process: using an approximation function, a candidate to nearest neighbour is selected and its distance to the sample is computed. Then, if it is closer to the sample than the current NN, it becomes the current NN and the training set is pruned so that all the prototypes that are outside an hypersphere centered in the sample with radius d_{nn} (the distance to the current NN) are safely eliminated from the training set. The process continues until the training set is empty, and then the current NN will be the NN.

In this paper we propose a new classification rule that makes use of the prototypes that are selected in a standard 1-NN search as candidates to nearest neighbour by fast NN search algorithms. To illustrate the behaviour of this rule, several fast and widely known NN search algorithms have been extended with it, obtaining classification results similar to those of a k -NN ($k > 1$) classifier without the extra computational overhead. Also, previous work on approximate NN search for vector spaces has been extended to algorithms suitable for general metric spaces, and has been combined with the new classification rule.

The paper is structured as follows: the next section briefly describes the new classification rule, which is based on the approximation and elimination framework. Then, the approximate NN search for some (metric spaces) algorithms is outlined. The experiments section will show the results of the new rule when applied to various NN search algorithms in experiments with synthetic and real data, and also the results of combining the rule with approximate NN search. Finally, we will conclude and outline some future work.

2 The k -NSN classification rule

Many approximation and elimination search algorithms are based on the following idea: during preprocessing, a data structure is built to allow pruning of the training set. Then, during classification, a candidate to nearest neighbour is selected and stored, and its distance to the sample is computed. This distance is then used to prune the training set (using the data structure) and maybe to select a new candidate. This process ends when all the training set has been pruned or selected. Extending such an algorithm to find the k -NN is usually simple: each time a distance is computed (step (a) in figure 1), it is stored (along with the prototype) in a sorted array that holds the k -NN found so far. Then, the distance used to prune the training set is the distance to the k th nearest neighbour found so far, instead of the distance to the current NN (d_{nn} in step (b) of figure 1). This involves less pruning and more distances to compute, which derives in an additional computational overhead, always dependent on the value of k .

In this paper we propose a simple but powerful extension for any approximation and elimination based NN search algorithm: when looking for the nearest neighbour, each prototype selected and its distance to the sample are stored in a sorted array, as for the k -NN search. However, the distance used to prune the training set is d_{nn} , so that the number of distances (and thus the computational effort) is the same as for a standard NN search. The prototypes stored are called the k nearest selected neighbours (k -NSN), and they are not exactly the k -NN. When the search finishes,

the sample is classified by majority voting using these neighbours (which include the nearest neighbour), as in the k -NN rule.

This technique can be considered a new classification rule (the k -NSN rule) which requires very little computational effort over a NN search (storing the k nearest selected neighbours)¹, and, as we shall see in a following section, it achieves classification results very similar to those of the k -NN rule. Also, if this rule is applied to an exhaustive NN search it yields the k -NN rule. The rule raises up as an extension of previous work on the LAESA algorithm [10, 11]. Given that classification time does not (highly) depend on the value of k , one may increase k as desired to improve classification rates; however, as also happens with the k -NN rule, from a certain value of k the rates start to worsen.

3 Approximate NN search in metric spaces

There are a number of real tasks for which finding exactly the NN (even using a fast NN search algorithm) may become too slow; a number of approximate NN search algorithms [12, 13, 14, 15] have been proposed to face these tasks, yielding slightly worse classification rates but obtaining much lower classification times.

However, these algorithms are usually based on vector spaces of representation, and this feature limits its range of application in Pattern Recognition tasks. Moreover, in some real tasks where a string or tree represents an object (and thus usually the string or tree edit distance is used), classification times are much higher than in vector space tasks. In this work we have extended some ideas from previous works on approximate NN search in vector spaces to algorithms suitable for general metric spaces: Fukunaga and Narendra's [2], AESA [5], LAESA [10] and TLAESA [16].

In a widely known implementation of approximate NN search by Arya and Mount [12], a priority queue is used in a kd-tree to store the nodes which the search algorithm has still to visit. The key for the queue is some kind of lower bound of the distance from the node to the sample, and the node with the minimum key is the first to be extracted from the queue.

We have tested a similar idea with the Fukunaga and Narendra's algorithm, and with TLAESA (both tree-based algorithms): when a non-leaf node is visited, its children are stored in the queue using a key, m , which is a lower bound of the

¹The simplest implementation is to insert the new pair distance/prototype in a sorted array of k pairs, if the distance is lower than the last one in the array. The extra time complexity over the NN search is $O(ck)$, where c is the number of computed distances. Although it is possible to reduce this time complexity with a heap, this overhead is almost negligible when compared to the overhead of computing c distances.

distance of the node to the sample². In each step of the algorithm, the node with the minimum key m is extracted from the queue, and the algorithm finishes when the following condition holds:

$$m > d_{nn}$$

where d_{nn} is the distance of the current nearest neighbour to the sample. For an approximate search, the condition has been changed to:

$$m \cdot (1 + \epsilon) > d_{nn}$$

where ϵ is a parameter to tune the search: the higher the value of ϵ , the faster the search, but the higher the error rate. The optimum value of ϵ is a trade-off between classification time and allowable increase in the error rate, and should be determined for each classification task.

In the Fukunaga and Narendra’s algorithm, any non-leaf node p has a representative M_p and a radius r_p . When a node is visited, the distances of the representatives of its children to the sample are computed and stored. Given a child p , the expression:

$$d(x, M_p) - R_p$$

is a (pessimistic) lower bound of the actual distance of the prototypes contained in p to the sample. Thus, if the child is not eliminated by the elimination condition of the algorithm (from which is derived the expression for the lower bound), it is stored in the queue along with the lower bound as the key.

The TLAESA algorithm does not compute any distances when visiting a non-leaf node; instead, it uses a lower bound of the distance from the representative M_p to the sample, $G[M_p]$, which is computed in the following way:

$$G[M_p] = \max_{b \in B} |d(x, b) - d(b, M_p)|$$

where B is a subset of prototypes called *base prototypes* (see [10, 16] for the details). The distances from each $b \in B$ to the sample are computed in the first step of the classification phase, and the distances $d(b, M_p)$ are computed (and stored) prior to classification, in a preprocessing step. The extension of this algorithm for approximate NN search has been done in a way very similar to that in the Fukunaga and Narendra’s algorithm. The key for each node, which is also a lower bound of the distance of all the prototypes in the node to the sample, is:

$$G[M_p] - R_p$$

²In the work by Arya and Mount, only the unvisited child (in a binary tree like the kd-tree) is stored in the queue.

so that the lower bound $G[M_p]$ is used instead of $d(x, M_p)$.

The AESA and LAESA algorithms are not based on trees, and thus a different scheme has been used. Both algorithms also compute a lower bound of the distance of each prototype to the sample; in the case of LAESA, the lower bound is computed exactly as in the TLAESA algorithm (in fact, the TLAESA algorithm is derived from the LAESA algorithm). In the AESA algorithm, a lower bound of the distance is also computed, but in a slightly different way (as if all the prototypes were base prototypes). In each step of the search phase of the AESA or LAESA the algorithm, the prototype p whose lower bound $G[p]$ is the minimum is selected as a candidate to NN; whenever

$$G[p] > d_{nn}$$

the algorithm finishes. For approximate NN search, this condition has to be changed into this one:

$$G[p] \cdot (1 + \epsilon) > d_{nn}$$

No further changes are needed in both algorithms. This kind of approximate search with these two algorithms is very similar to the search using a certain looseness [17].

4 Experiments with the k -NSN rule

Several series of experiments have been performed in order to test the application of the k -NSN rule to various fast NN search algorithms (see table 1). All these algorithms fit in an approximation and elimination framework, and all are suited for general metric spaces except kd-tree, which requires point coordinates. The algorithms of AESA family (AESA, LAESA, and TLAESA) focus on reducing the number of distance computations, thus are best suitable for expensive distances. The vp-tree and GNAT were developed to face large training sets and/or high dimensionality of data, and thus the number of distance computations is important but it is not its main goal.

Two sets of experiments have been performed: first, a set of synthetic data experiments to test the performance of the rule in a widely known environment. Second, several tests have been performed with a real data task, human chromosome classification. In both cases the main goal was to study the error rates of these algorithms using the k -NSN rule and to compare them with the k -NN error rates.

4.1 Experiments with synthetic data

For these experiments we have generated Gaussian data from 8 classes of dimensions 10 and 20 using the algorithm for generating clustered data in [18]. Tests have been

ALGORITHM	Author(s)
kd-tree	Friedman <i>et al.</i> [3]
FN75	Fukunaga and Narendra [2]
vp-tree	Yianilos [4]
AESA	Vidal [5]
LAESA	Micó <i>et al.</i> [10]
TLAESA	Micó <i>et al.</i> [16]
GNAT	Brin [6]

Table 1: Fast NN search algorithms which have been extended with the k -NSN rule.

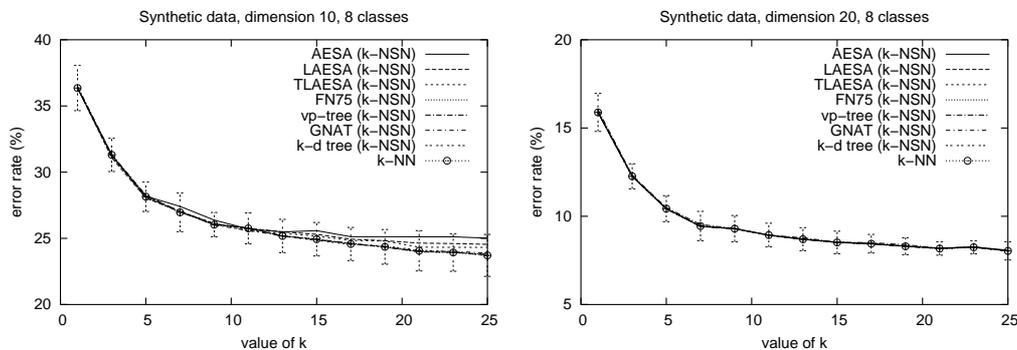


Figure 2: Comparison between k -NN and k -NSN classifiers, with synthetic data of dimensions 10 and 20. The error bars are plotted only for the k -NN rule and correspond to a 95% confidence interval.

performed using 10 different pairs of training and test sets, with 4096 and 1024 prototypes respectively. The plots compare the error rate of the k -NSN and k -NN rules as the value of k increases (figure 2). These results show that k -NSN and k -NN error rates are very similar (and are almost the same for dimension 20), and are better than those of an NN classifier.

Although the definition of the k -NSN rule assures that the number of distance computations remains the same as in the NN rule, an experiment has been developed to verify it and also to show that in the k -NN rule the number of distances increase with the value of k . Figure 3 shows the results for the Fukunaga and Narendra's algorithm, named FN75 in the plots.³

³The results with all the other algorithms are similar and are not showed for brevity.

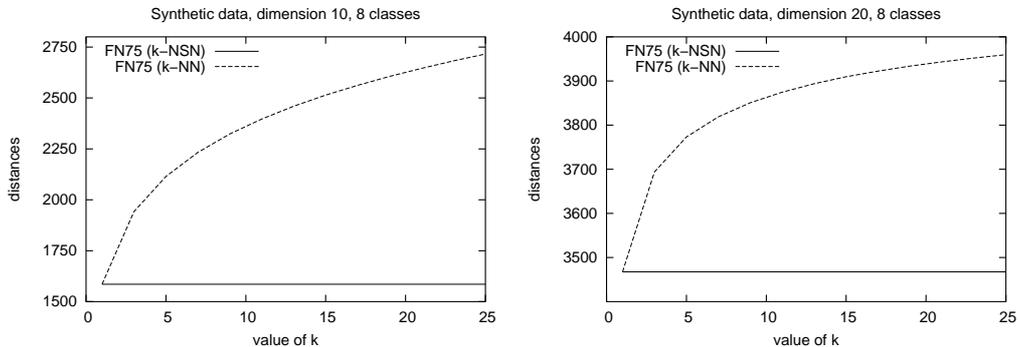


Figure 3: Average distance computations of k -NSN and k -NN rules using the Fukunaga and Narendra’s algorithm (FN75), with synthetic data of dimensions 10 and 20.

4.2 Experiments with real data

The real data experiments have been developed for a human chromosome classification task [19, 20, 21]. The chromosome database contains 4400 samples coded as strings, and the edit or Levenshtein distance [22] has been used for this task; the kd-tree makes use of the coordinates of the prototypes, so it has not been tested with this database. The database is splitted 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. Figure 4 shows the average error rates of k -NN and k -NSN classifiers as the value of k increases. There is a parameter for LAESA and TLAESA (see [10, 16] for more details), the number of *base prototypes*, which has been set to 40.

The average number of distance computations and classification times are plotted in figures 5 and 6. Each individual plot compares k -NN and k -NSN values as the value of k increases. The results confirm that the number of distance computations and the average classification time per test sample of the k -NSN rule does not depend on the value of k ; also, the plots show the main advantage of the k -NSN rule over the k -NN rule: whereas in the k -NN rule the time performance worses as the value of k increases, the k -NSN rule maintains the same time as the NN rule, while obtaining almost the same error rates than the k -NN rule.

4.3 How many of the k -NSN are among the k -NN?

The k -NSN rule obtains error rates very close to those of the k -NN, and one may think that this is due to the fact that many of the k -NSN are in fact among the

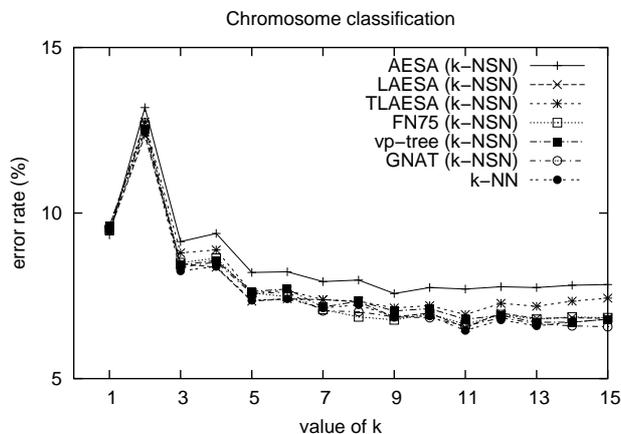


Figure 4: Error rate of k -NN and k -NSN rules in chromosome classification.

k -NN. Another possibility is that the k -NSN, even not matching exactly with the k -NN, are near at the same distance to the sample.

In order to study this question we have developed two experiments: in the first one, using synthetic data from varying dimensions (from 5 to 50) and taking two measures: the percentage of matching, that is, how many of the k -NSN are among the k -NN, and the relation between the distance of the last k -NSN and the last k -NN. The experiment has been repeated with 10 different pairs of training and test sets of 4096 and 1024 prototypes respectively. Figures 7 and 8 show a plot and a table with the results, which show that when dimension increases both the percentage of matching and the relation get close to the optimum (100% and 1). This is why the k -NSN error rates are almost the same as the k -NN rates when data dimension increases.

Our second experiment on this question was far more ambitious. We thought that if the percentage of matching clearly closes or reaches 100% when increasing the training set size, we could state that, in the limit (when the training set size closes infinite), the k -NSN match exactly the k -NN, and thus the k -NSN rule has the same statistical properties as the k -NN rule, i.e., that its error rate is bounded by as much 2 times the Bayes error [23]. The second experiment tries to see if this hypothesis holds. In this case, the training set size was varying from 1024 to 65536, with dimension 10, and the test set had 1024 prototypes. The results are plotted in figure 9, and they seem to prove that our hypothesis was not true.

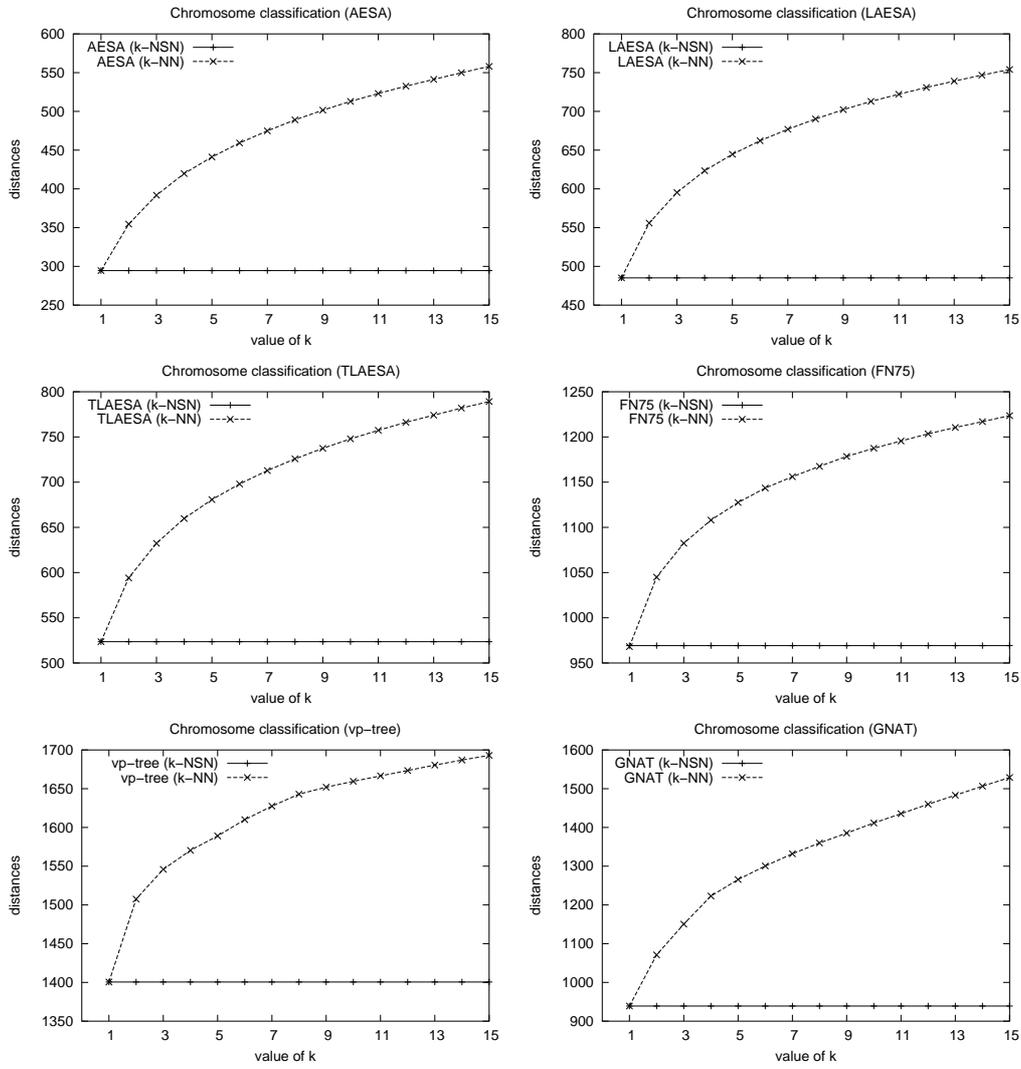


Figure 5: Comparison between k -NN and k -NSN average distance computations in the chromosome classification task.

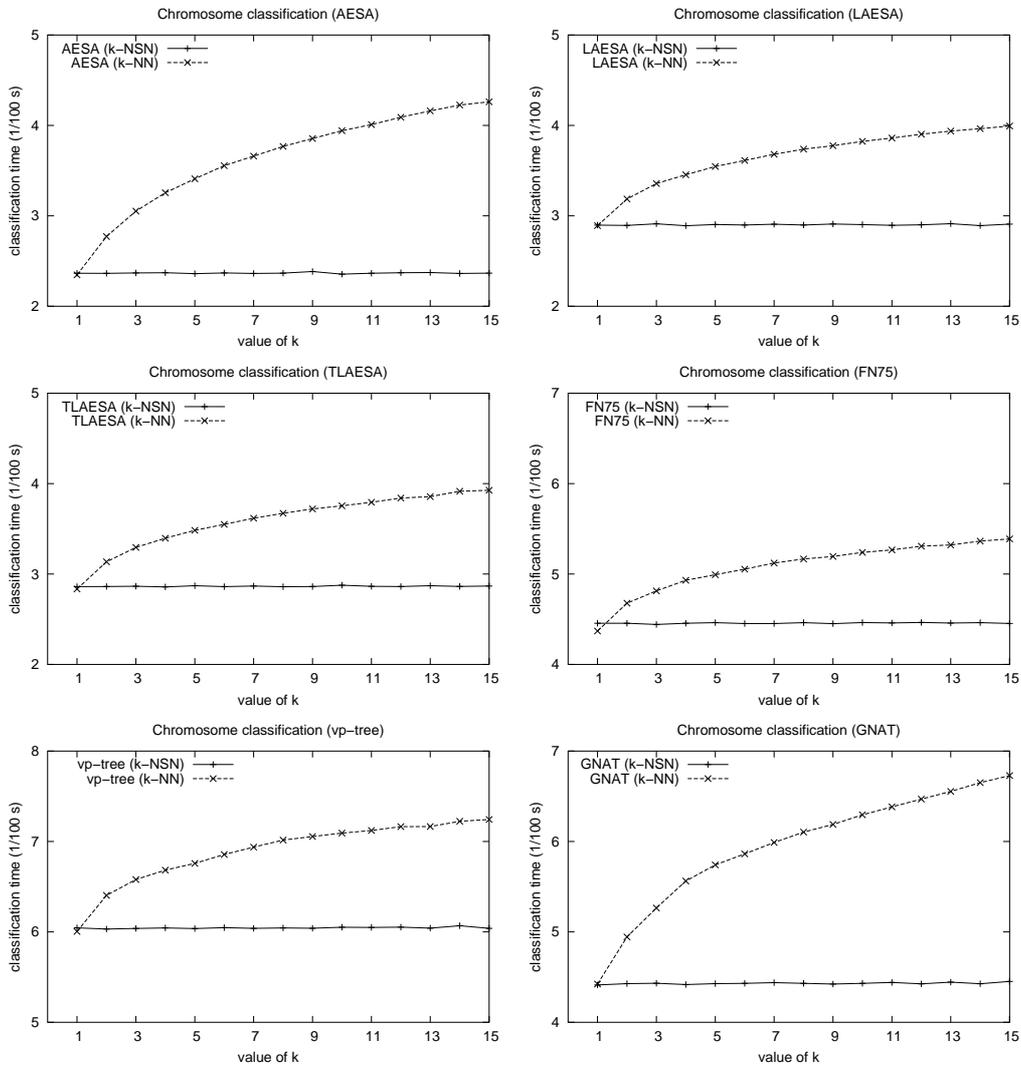
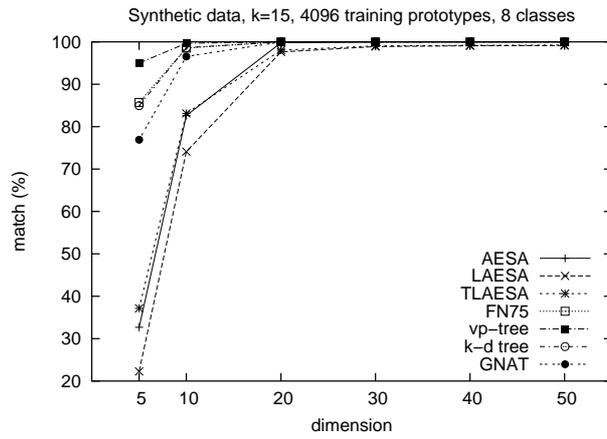
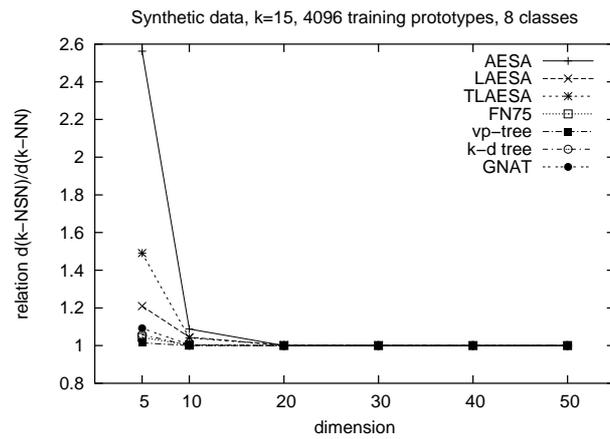


Figure 6: Comparison between k -NN and k -NSN average classification time per sample in the chromosome classification task.



Dim	LAESA	TLAESA	AESA	GNAT	FN75	kd-tree	vp-tree
5	22.26	37.13	32.66	76.91	85.71	84.94	95.00
10	74.09	83.13	82.61	96.55	98.55	98.64	99.69
20	97.61	98.03	99.78	99.98	100	100	100
30	98.91	99.04	100	100	100	100	100
40	99.07	99.19	100	100	100	100	100
50	99.15	99.26	100	100	100	100	100

Figure 7: Percentage of the k -NSN that are among the k -NN, for synthetic data of various dimensions.



Dim	LAESA	TLAESA	AESA	GNAT	FN75	kd-tree	vp-tree
5	1.21	1.49	2.56	1.09	1.04	1.06	1.02
10	1.05	1.04	1.09	1.01	1.00	1.00	1.00
20	1.00	1.00	1.00	1	1	1	1
30	1.00	1.00	1	1	1	1	1
40	1.00	1	1	1	1	1	1
50	1	1	1	1	1	1	1

Figure 8: Relation between the distance to the sample of the k th NSN and the distance of the k th NN, for synthetic data of various dimensions. The value 1.00 indicates that is slightly greater than 1, but not exactly 1.

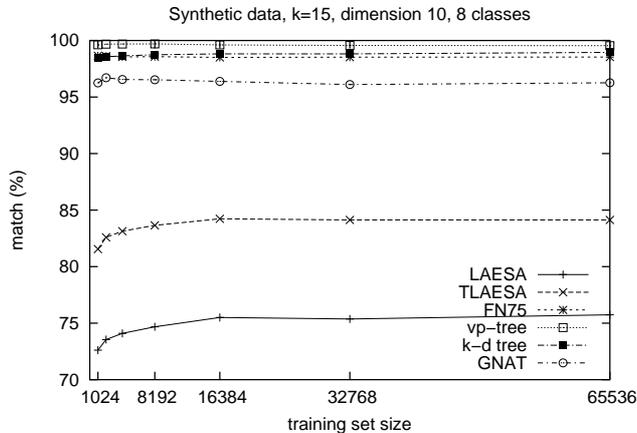


Figure 9: Percentage of the k -NSN that are among the k -NN, for synthetic data with increasing training set size.

5 Experiments with approximate NN search

The goal of approximate NN search is to reduce classification time, maybe slightly increasing error rates. In order to test the approach proposed in section 3, several experiments with both synthetic and real data have been developed. For the synthetic data, the objective was to compare the error rates and distance computations of the algorithms for general metric spaces with the algorithm by Arya et al. [12], in whose ideas we inspired to develop our technique. In the case of real data tasks, we have developed experiments with the chromosome classification task, using the string edit distance.

5.1 Experiments with synthetic data

The implementation the Arya and Mount algorithm was taken from the ANN software package [24], and the experiment was developed with dimension 10 data, using 4096 prototypes for training and 1024 for test. As in previous experiments, 10 different pairs of train/test sets were used. The value of k was set to 25, and in both cases the classification rule was the k -NN rule. Figure 10 plots the error rates and the distances computed by all the algorithms with increasing value for ϵ . The results show that ANN error rates are only beaten by the Fukunaga and Narendra's algorithm, but it computes a higher number of distances; however, the ANN package uses kd-trees, which compute a high number of partial distances (which have

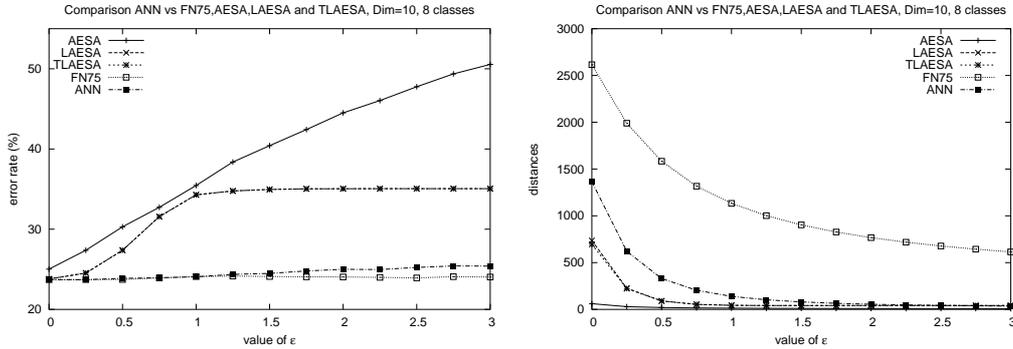


Figure 10: Comparison of error rates (left) and distances computed (right) for the ANN, Fukunaga and Narendra’s algorithm, AESA, LAESA and TLAESA.

not been accounted in this experiment). The AESA family algorithms seem to be competitive only for very low values of ϵ , as its error rates increase very quickly with the value of ϵ .

5.2 Experiments with real data

Approximate NN search is specially indicated when the distance is very time consuming, as in the case of the string edit distance in the chromosome classification task mentioned before. Figure 11 plots the results for a simple 1-NN search using various values for ϵ ; dotted lines represent error rates, whereas non-dotted lines represent average classification time per sample. As can be appreciated in the figure, the Fukunaga and Narendra’s algorithm seems to be the best if we exclude the AESA algorithm, that has a quadratic spatial complexity that limits its applicability. The LAESA and TLAESA results are almost equal due to the fact that both use the same lower bound of the distance from a prototype to the sample.

For the chromosome task the optimum value of k is 11 (see figure 4), and thus we tested the approximate search with $k = 11$ on the Fukunaga and Narendra’s algorithm, using both the k -NN and k -NSN rules. The results are plotted in figure 12, and show that using approximate search in combination with the k -NSN rule produces a lower classification time, but with an error rate that is slightly higher than k -NN rate. The important point is that the behaviour of the k -NSN rule remains the same with approximate search than with standard search.

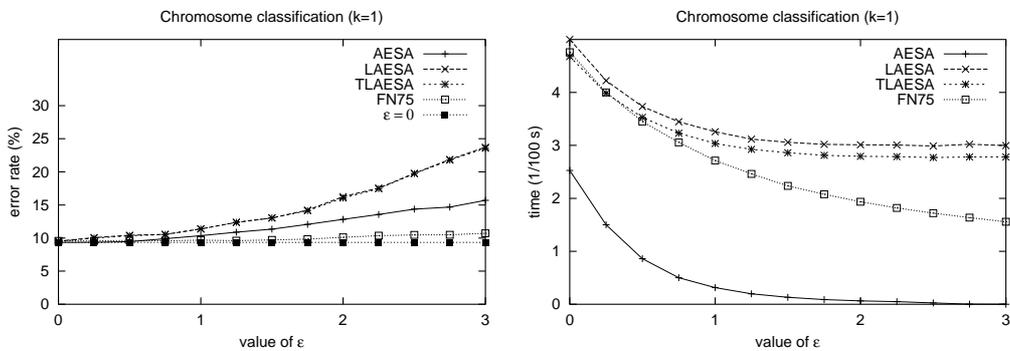


Figure 11: Error rates (left) and classification times per sample (right) in the chromosome classification task, using approximate NN search.

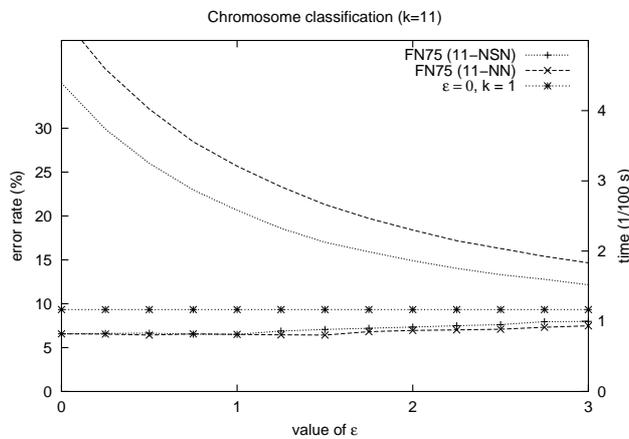


Figure 12: Error rates (lines with points) and classification times per sample (lines without points) in the chromosome task, with the Fukunaga and Narendra's algorithm using both the k -NSN and the k -NN rules, for $k = 11$.

6 Conclusions and future work

A new NN based classification rule (the k -NSN rule) has been developed and tested with various well known fast NN search algorithms, which fit into the approximation and elimination framework: kd-tree, Fukunaga and Narendra's, vp-tree, GNAT. The rule has also been tested with the algorithms of AESA family, which also fit in the approximation and elimination framework.

The experiments show that classification results similar to those of the k -NN rule are obtained using this rule with very little extra computational effort with respect to a NN classifier. Whenever a fast approximation and elimination NN search algorithm is applicable, it may be easily modified to classify using the k -NSN rule and thus it may obtain error rates lower than those of NN, without the extra overhead of searching for the k -NN. Moreover, the time performance of k -NSN classifiers does not depend on the value of k , and the error rates decrease (and get closer to those of the k -NN rule) as the dimensionality increases. The k -NSN rule may be an alternative for the k -NN rule when classification time is an important question in a classification task; also, it may be employed to determine the optimum value for k in the design of a classifier, even if the k -NN rule is finally chosen.

In addition to this rule, previous work on approximate NN search for vector spaces has been extended to algorithms suitable for general metric spaces, and it has been combined with the k -NSN rule, yielding very interesting results for real tasks.

There is still a lot of work to do to explore the possibilities and range of application of the k -NSN rule and approximate NN search. As for the future, we plan to:

- study the evolution of k -NSN error rates as the value of k become higher than those tested in this work, and compare them with k -NN,
- test the performance of the k -NSN rule as the dimensionality or the number of classes increase, and
- apply the k -NSN rule to other approximation-elimination NN search algorithms.
- extend approximate NN search to other algorithms not based on vector spaces, and study its performance in combination with the k -NSN rule.

References

- [1] R. Duda and P. Hart. *Pattern Recognition and Scene Analysis*. Wiley, 1973.
- [2] K. Fukunaga and M. Narendra. A branch and bound algorithm for computing k -nearest neighbors. *IEEE Transactions on Computing*, 24:750–753, 1975.
- [3] J.H. Friedman, J.L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3:209–226, 1977.
- [4] P.N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, 1993.
- [5] E. Vidal. New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (AESAs). *Pattern Recognition Letters*, 15:1–7, 1994.
- [6] S. Brin. Near neighbor search in large metric spaces. In *Proceedings of the 21st VLDB Conference*, pages 574–584, 1995.
- [7] S. Nene and S. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):989–1003, 1997.
- [8] E. Chavez, G. Navarro, R.A. Baeza-Yates, and J.L. Marroquin. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.
- [9] V. Ramasubramanian and K.K. Paliwal. Fast nearest-neighbor search algorithms based on approximation-elimination search. *Pattern Recognition*, 33:1497–1510, 2000.
- [10] L. Micó, J. Oncina, and E. Vidal. A new version of the nearest neighbour approximating and eliminating search algorithm (AESAs) with linear preprocessing-time and memory requirements. *Pattern Recognition Letters*, 15:9–17, 1994.
- [11] F. Moreno-Seco, L. Micó, and J. Oncina. A modification of the LAESA algorithm for approximated k -nn classification. *Pattern Recognition Letters*, 24(1–3):47–53, 2003.

- [12] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45:891–923, 1998.
- [13] P. Indyk and R. Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. In *Proceedings of the 30th ACM Symposium on Theory of Computing*, pages 604–613, 1998.
- [14] K.L. Clarkson. Nearest neighbor queries in metric spaces. *Discrete Computational Geometry*, 22(1):63–93, 1999.
- [15] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proceedings of 29th ACM Symposium on Theory of Computing*, pages 599–608, 1997.
- [16] L. Micó, J. Oncina, and R. C. Carrasco. A fast branch and bound nearest neighbour classifier in metric spaces. *Pattern Recognition Letters*, 17:731–739, 1996.
- [17] E. Vidal, F. Casacuberta, and H. Rulot. Is the dtw “distance” really a metric? an algorithm reducing the number of dtw comparisons in isolated word recognition. *Speech Communication*, (4):333–344, 1985.
- [18] A.K. Jain and R.C. Dubes. *Algorithms for clustering data*. Prentice-Hall, 1988.
- [19] C. Lundsteen, J. Phillip, and E. Granum. Quantitative analysis of 6985 digitized trypsin G-banded human metaphase chromosomes. *Clinical Genetics*, 18:355–370, 1980.
- [20] E. Granum, M.G. Thomason, and J. Gregor. On the use of automatically inferred Markov networks for chromosome analysis. In C. Lundsteen and J. Piper, editors, *Automation of Cytogenetics*, pages 233–251. Springer-Verlag, Berlin, 1989.
- [21] E. Granum and M.G. Thomason. Automatically inferred Markov network models for classification of chromosomal band pattern structures. *Cytometry*, 11:26–39, 1990.
- [22] R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1):168–173, 1974.
- [23] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 1990.

- [24] D.M. Mount and S. Arya. Ann: A library for approximate nearest neighbor searching, 1997. url: <http://www.cs.umd.edu/~mount/ANN>.