Pattern Recognition 35 (2002) 2791–2799

Efficient prototype reordering in nearest neighbor classification

Sanghamitra Bandyopadhyay[a, *], Ujjwal Maulik[b]

[a]*Machine Intelligence Unit, Indian Statistical Institute, 203 B.T. Road, Calcutta 700 035, India*
[b]*Department of Computer Science and Technology, Government Engineering College, Kalyani, India*

Received 18 June 1999; received in revised form 17 April 2001; accepted 18 July 2001

Abstract

Nearest Neighbor rule is one of the most commonly used supervised classification procedures due to its inherent simplicity and intuitive appeal. However, it suffers from the major limitation of requiring n distance computations, where n is the size of the training data (or prototypes), for computing the nearest neighbor of a point. In this paper we suggest a simple approach based on rearrangement of the training data set in a certain order, such that the number of distance computations is significantly reduced. At the same time, the classification accuracy of the original rule remains unaffected. This method requires the storage of at most n distances in addition to the prototypes. The superiority of the proposed method in comparison to some other methods is clearly established in terms of the number of distances computed, the time required for finding the nearest neighbor, number of optimized operations required in the overhead computation and memory requirements. Variation of the performance of the proposed method with the size of the test data is also demonstrated.

Keywords: Pattern recognition; NN rule; Neighborhood computation; Supervised classification

1. Introduction

Classification of data forms an important component of all pattern recognition systems where the features are passed on to the classifier that evaluates the incoming information and makes a final decision. This phase basically establishes a transformation rule between the features and the classes. One commonly used non-parametric transformation procedure is the nearest neighbor (NN) rule [1–3]. In addition to pattern classification, the NN rule is also widely used in several areas such as non-parametric density estimation and data compression using vector quantization. The popularity of this classification technique is because of its intuitive appeal, simplicity and effectiveness. However, it suffers from one major limitation of requiring a large number of distance computations for identifying the nearest neighbor of a point.

Several attempts have been made in the past to reduce the number of distance calculations and hence improve the efficiency of the NN rule by incorporating a preprocessing step on the prototype set (or the training data points) [4–12]. In Ref. [4] the prototypes are ordered based on the values of one of the coordinates. For each test point, the prototypes are examined in the order of their projected distance from the test point on the sorted coordinate. When the projected distance becomes larger than the distance, in full dimensionality, to the closest point of those prototypes already examined, no more prototypes need to be considered. The predicted number of distance calculations for computing the nearest neighbor in case of bivariate normal data in two dimensions is 36 for 1000 prototypes while simulations have shown this to be equal to 24. In Ref. [5] a clustering technique for hierarchically decomposing the prototype set is combined with a branch-and-bound technique to achieve substantial reduction in computations. It was found that on an average 61 distances had to be computed for finding the nearest neighbor of a point among 1000 prototypes.

* Corresponding author.
E-mail addresses: sanghami@isical.ac.in (S. Bandyopadhyay), ujjwal_maulik@kucse.wb.nic.in (U. Maulik).

Yunck [6] has projected the prototypes onto each axis and defined a hypercube around the point to be classified. By changing the size of this hypercube, the one containing at least one point can be identified. It is found that in case of the Euclidean metric used in two dimensions with 1000 uniformly distributed points, the effective number of distance calculations required to find the nearest neighbor is approximately five, while it goes up to 493 for ten-dimensional space. Here, although a significant gain is achieved in terms of the computational efficiency, the storage requirement is found to increase almost thrice. Sethi [7] proposed another method of fast recognition of nearest neighbors by computing the distances from three reference points. However, the major drawback of this algorithm is that it does not retain the classification accuracy of the NN rule. In other words, this method actually utilizes a classification rule (slightly) different from the NN rule.

Vidal [8] proposed an approximating and eliminating search algorithm (AESA) for finding the nearest neighbors that is independent of the size of the data set. The method uses the metric properties of the given distance, and does not assume the data to be structured into any vector space. It is shown that the number of distance computations for finding the nearest neighbor is < 4 in two dimensions and < 60 in ten dimensions. However, this method had quadratic memory and preprocessing time requirements and involved significant overhead. Farago et al. [9] besides providing a theoretical analysis of AESA, also proposed a modification of the algorithm thereby requiring only $n(d+1)$ distances to be stored (as opposed to n^2 in case of AESA, where n is the size of the training data and d is the number of dimensions). They further propose a classification rule which provides the same asymptotic error probability as that of the NN rule, and calculates only $d+1$ distances deterministically.

Vidal in Ref. [10] subsequently proposed an enhancement of AESA, that uses branch and bound scheme in the selection or approximating criterion which resulted in three improvements. It was significantly cheaper to compute, reduced the overhead and also improved the performance of AESA slightly. Mico et al. [11] also suggested another linear AESA (LAESA) which required only linear memory and preprocessing time. They studied the performance of the LAESA through some simulation experiments and showed that it overcame the quadratic bottleneck of the AESA. However, it required about 1.5 times more distance computations than the AESA.

In this article, we propose a very simple preprocessing step of the set of prototypes that will ensure that the nearest neighbors can be obtained with significantly smaller number of distance computations, while retaining the exact performance of the NN rule. The preprocessing step essentially reorders the prototypes according to their distances from a fixed point in the feature space usually chosen as one extreme point of the hypervolume surrounding the prototypes. The distances are also stored along with the prototypes. This entails a maximum increase in memory requirement from $n \times d$ (for the original set of prototypes) to $n \times (d+1)$. Subsequently, the nearest neighbor of a test point is identified by growing a hypercube around this point. At the same time, the information about the distances of the prototypes from the fixed point is taken into account so as to reduce the number of prototypes to be considered.

The performance of this nearest neighbor algorithm using reordering of the prototype set (referred to as PR–NN procedure) is studied using a number of simulation experiments for a uniformly distributed data set. Since the approximating and eliminating search algorithm (AESA) of Vidal [10] and an improved version (LAESA) [11], to the best of our knowledge, requires quite few distance computations with respect to other techniques available in the literature, we have compared our method to theirs in terms of the number of distance computations, time required for finding the nearest neighbor, number of operations involved in overhead computation and memory requirements. In a part of the investigation, experiments pertaining to computation of the k ($k \geqslant 1$) nearest neighbors of a point are also carried out and the results are included.

2. Nearest neighbor identification procedure

In this section we first outline the basic NN procedure followed by a description of the PR–NN rule in detail.

2.1. NN rule

Let us consider a set of n labeled pattern $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ belonging to one of the classes $C_1, C_2, \ldots, C_k$. The NN classification rule assigns an unlabelled pattern $\mathbf{y}$ to the class of its nearest neighbor $\mathbf{x}_i \in \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ such that

$$D(\mathbf{x}_i, \mathbf{y}) = \min_l \{D(\mathbf{x}_l, \mathbf{y})\}, \quad l = 1, 2, \ldots, n, \tag{1}$$

where D is any distance measure defined over the feature space.

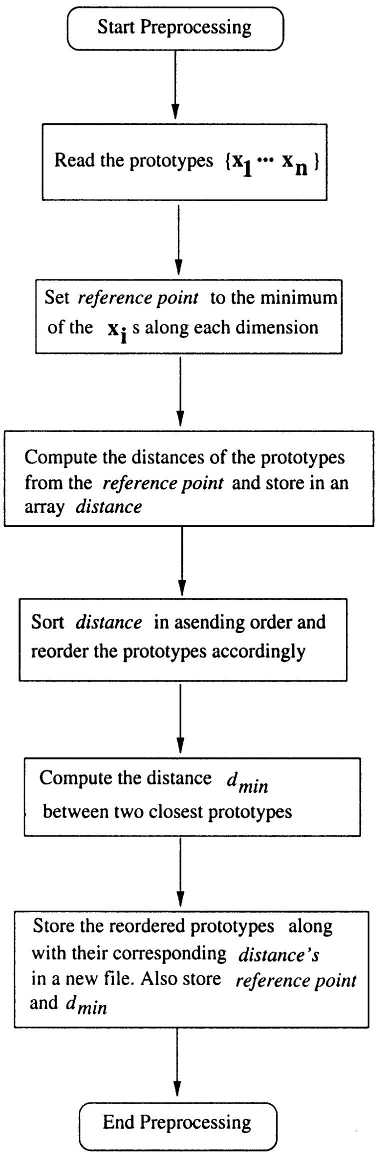
Since the aforesaid scheme employs the class label of only the nearest neighbor to $\mathbf{y}$, this is known as the NN rule. The details of the NN rule along with the probability of error are available in Refs. [1–3].

2.2. PR–NN procedure

The PR–NN procedure comprises two primary phases—preprocessing and neighborhood identification. These are now detailed below.

2.2.1. Preprocessing

The block diagram of the preprocessing phase of the PR–NN procedure is provided in Fig. 1. In this phase, the training data (or the sample prototypes) are read and minimum along each dimension, m_i ($i = 1, 2, \ldots, d$, where d is the dimension of the feature space), is computed. The point



Start Preprocessing
Read the prototypes {x1 ··· xn}
Set reference point to the minimum of the xi s along each dimension
Compute the distances of the prototypes from the reference point and store in an array distance
Sort distance in asending order and reorder the prototypes accordingly
Compute the distance dmin between two closest prototypes
Store the reordered prototypes along with their corresponding distance's in a new file. Also store reference point and dmin
End Preprocessing

Fig. 1. Preprocessing.

having the coordinate $(m_1, m_2, \ldots, m_d)$ is taken as the *reference point* (or $\mathbf{x}_c$). Fig. 2 shows an example of reference point in three dimensional space. Subsequently, the distance $(d_i,\ i = 1, 2, \ldots, n)$ of each point from the *reference point* is calculated. These distances, along with the corresponding prototypes, are sorted (in ascending order) and stored. Finally, the distance between two closest points (d_{min}) in the data set is computed which is used in the subsequent neighborhood computation phase to fix the width of the surrounding hypercube around the test point. The value of d_{min} as well as the *reference point* are also stored. If there are n prototypes in d dimensional space then the amount of memory required for storing all this information will be

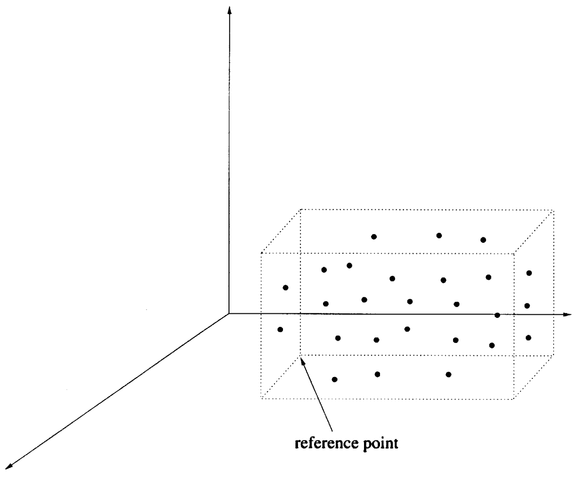


Fig. 2. Example of a *reference point*.

$(n + 1) \times (d + 1)$, out of which $n \times d$ is required for storing the prototypes themselves.

One may note that the complexities for computing the distances, sorting and computing d_{min} are $O(n)$, $O(n \log(n))$ and $O(n^2)$, respectively. Thus the overall computational complexity of the preprocessing phase is $O(n^2)$. Since the preprocessing has to be carried out only once for a given set of prototypes, its computational complexity does not play a major role. The memory requirement that is linear in the number of prototypes is a major advantage of this technique. One may also note in this context that the requirement for computing d_{min} may be relaxed; thereby reducing the complexity of the preprocessing phase to $O(n \log(n))$.

2.2.2. Neighborhood identification

The different steps of the neighborhood identification procedure is given below.

Step 1: Let $\mathbf{y} = (y_1, y_2, \ldots, y_d)$ be the sample whose nearest neighbor has to be computed. Let $N_y, X, \mathbf{x_c}$ and d_{min} be the set containing the neighbors of $\mathbf{y}$, set of reordered prototypes, reference point and distance between the two closest prototypes, respectively.

Step 2: Set $\delta = 1$, $d_y = distance(\mathbf{y}, \mathbf{x_c})$ and $N_y = N'_y = \emptyset$, where N'_y is an auxiliary data structure.

Step 3: Compute $\varepsilon = \delta \times d_{min}$.

Step 4: Set index I_l to the first prototype i in X, such that $d_i \geqslant d_y - \varepsilon$, (where d_i is the distance of the ith prototype from $\mathbf{x_c}$). Similarly, set index I_h to last prototype j in X, such that $d_j \leqslant d_y + \varepsilon$. Note that since X is sorted, the index settings can be done using binary search.

Step 5: Among the elements in between (and including) I_l and I_h, select only those elements whose coordinates lie in between $y_i \pm \varepsilon$, $i = 1, 2, \ldots, d$, (i.e., in the hypercube of width 2ε around $\mathbf{y}$). Add these elements to the set N'_y.

Step 6: If $N'_y = \emptyset$ increment δ by 1 and go to Step 3. Otherwise go to Step 7.

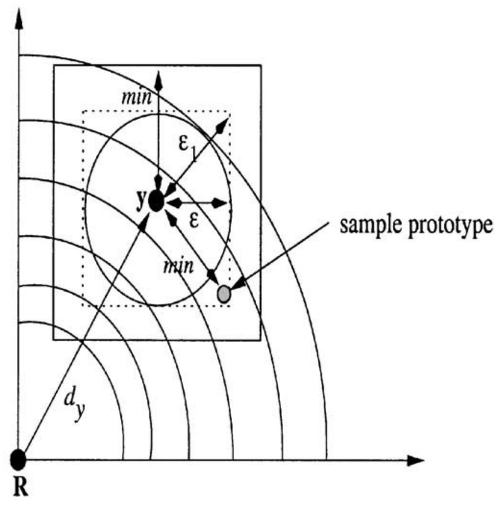


Fig. 3. Neighborhood computation process.

Step 7: For each $\mathbf{x}_i \in N'_y$, check whether its distance, $d_i^{\mathbf{y}}$, to $\mathbf{y}$ has already been computed. Otherwise compute $d_i^{\mathbf{y}}$. Also compute the minimum, *min*, of all $d_i^{\mathbf{y}}$ such that $i \in N'_y$, and add the corresponding element to N_y.

Step 8: If $min \leqslant \varepsilon$, then terminate with the nearest neighbor in N_y. Otherwise set $\varepsilon = min$, and go to Step 4.

Fig. 3 demonstrates the logic underlying the neighborhood determination process. It relies on the procedure of growing a hypersphere of diameter 2ε around the test point until it contains atleast one point from the training set. Since it is computationally intensive to check whether a training point lies in the hypersphere, the enclosing hypercube of sides 2ε is considered instead. Let the hypercube contain one training point in it (with the minimum distance from $\mathbf{y}$ being *min*) for some value of ε. At this point, let ε_1 be the distance of one corner point of the hypercube from $\mathbf{y}$ (see Fig. 3). If $min \leqslant \varepsilon$, then obviously this is the nearest neighbor. Otherwise, i.e., when $\varepsilon < min \leqslant \varepsilon_1$, we need to check only the hypersphere of radius *min* in order to find out the nearest neighbor of $\mathbf{y}$. This is done by actually setting $\varepsilon = min$ (rather than $\varepsilon = \delta \times d_{min}$ in Step 3) in Step 8 and going back to Step 4.

One may note that once the algorithm reaches Step 7 (which it is sure to reach at some point of time), then just one more iteration through the loop is required before the algorithm terminates. Moreover, in this outer loop, it will always find N'_y to be non-empty, and hence will not spend time any further time in the inner loop (Step 3 to Step 6).

Note: An extension of PR–NN procedure to the case of k nearest neighbors may be incorporated by maintaining an array $min[k]$ rather than just a scalar *min*, as done in Step 7 of the above algorithm. This array will be used to store the k minimum distances of the unknown point y from k prototypes which may then be included in the set N_y. In order to achieve this, an outer loop that will include Steps 4–8 of the algorithm needs to be added which will keep a check on whether k nearest neighbors have been already found or not.

3. Results

In this section, the effectiveness of the PR–NN technique is established through some simulation experiments. Its superiority over some other available methods is demonstrated in terms of the number of distances to be computed, the overall time requirement (excluding the preprocessing time) for finding the nearest neighbor, the number of optimized operations required for overhead computation, and the memory requirements. Note that preprocessing is done only once irrespective of the number of samples to be tested and therefore, it is not as important as the other factors. Different sizes of the training data set with varying dimensionalities are considered for this purpose. Variation of the performance of PR–NN with the size of the test set is also investigated. Results, in terms of the number of distance computations, of PR–NN when k nearest neighbors ($k = 1, 3, 5, 7$ and 9) are to be computed are reported at the end of this section.

3.1. *Performance of PR–NN*

The results presented in this section use training prototypes and test data drawn from a uniform distribution in the unit hypercube in d dimensional space. After having drawn a random set of n prototypes for training, an independent test set of 1000 prototypes is drawn from the same uniform distribution. The results are the average values obtained for these 1000 test points. The values of n and d considered for the experiments range from [100 to 10,000] and [2 to 10], respectively. In a part of the experiments, the variation of the performance of PR–NN with the size of the test set for fixed n (=1000) and for several values of d in the range 2–10 is demonstrated.

Fig. 4 shows the variation of the number of distance computations (N_{neigh}) with n for different values of d for both PR–NN and NN procedures. (Note that N_{neigh} is independent of d for the NN procedure.) As is evident from the figure, N_{neigh} is significantly less for PR–NN than that required by the NN rule. Moreover, the variation of N_{neigh} with n for a given d is comparatively small, indicating the stability of the proposed method. The variations of N_{neigh} with d for different fixed values of n are shown in Fig. 5. It is found that the maximum value of N_{neigh} when $n = 10,000$ and $d = 10$ is about 115, significantly less than what is required by the NN rule (i.e., 10,000).

It may be noted in this regard that the PR–NN procedure involves management of indices, and determination of prototypes which lie in the enclosing hypercube in addition to the computation of distances to the N_{neigh} neighbors. Although some sort of overhead is always involved in any procedure attempting to reduce the number of distance computations

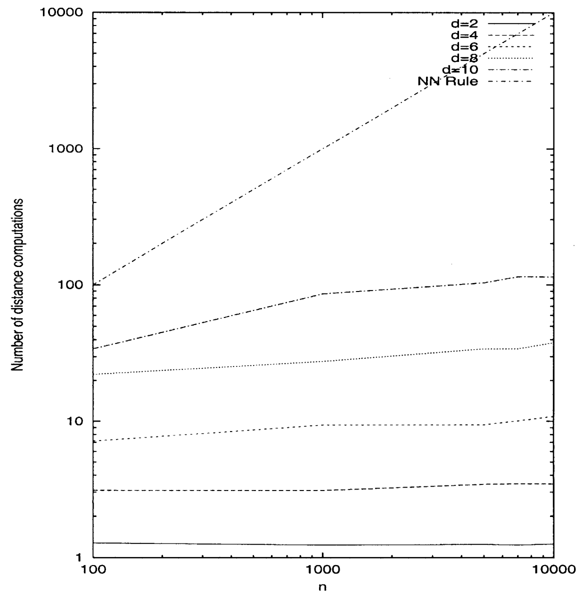


Fig. 4. Variation of number of distance computations with n for different values of d.

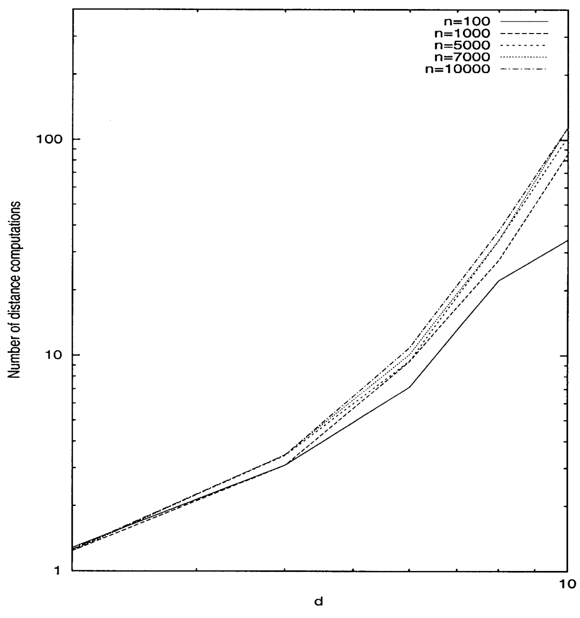


Fig. 5. Variation of number of distance computations with d for different values of n.

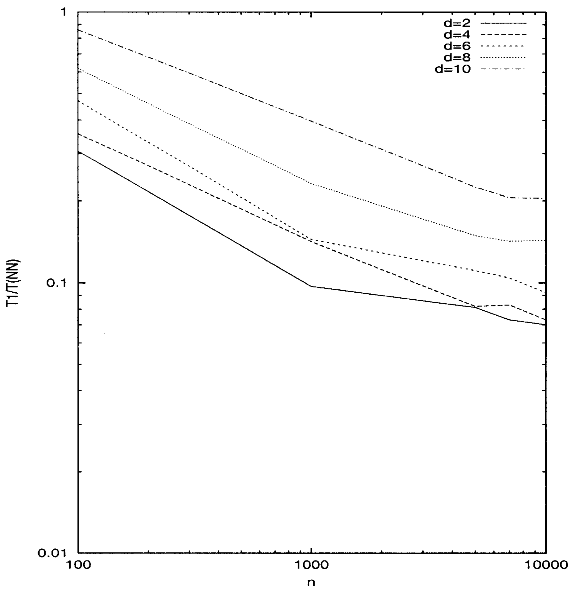


Fig. 6. Variation of ratio of time taken by PR–NN procedure to that by NN procedure with n for different values of d.

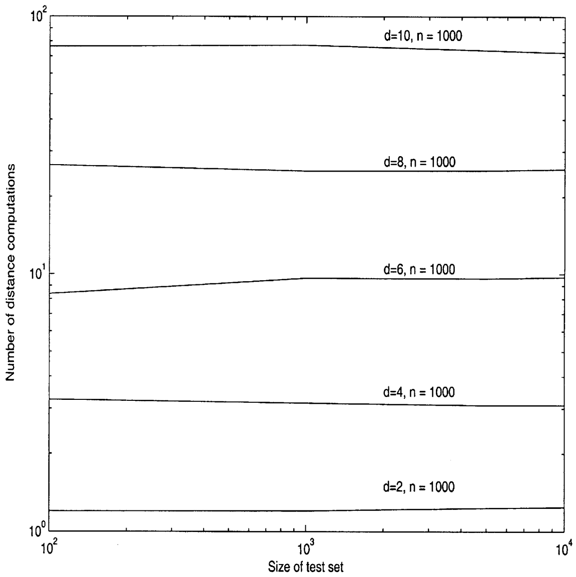


Fig. 7. Variation of number of distance computations with the size of the test set for different values of d and $n = 1000$.

[4–6,8,10,11], this gives rise to the issue of investigating the actual computation time required by the proposed procedure vis-a-vis the exhaustive NN-rule. Fig. 6 shows the variation of the ratio of time taken by the PR–NN procedure to that by the NN rule with n for different values of d. As can be seen from the figure, the time taken by PR–NN procedure is always much less than that taken by NN rule. Moreover, this ratio is found to decrease with increase in n, thereby indicating its applicability to larger data sets.

A study of the variation of the number of distances computed by PR–NN procedure with the size of the test set for fixed number of prototypes (=1000) is provided in Fig. 7 for $d = 2, 4, 6, 8,$ and 10. As seen from the figure, the number of distances computed, on an average, is relatively indepen-

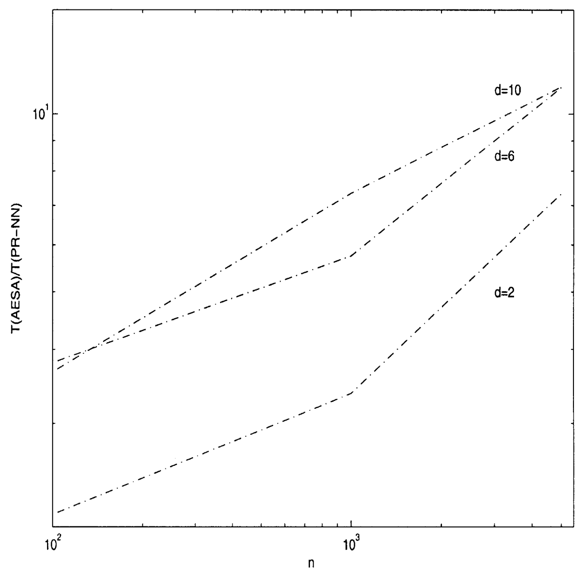


Fig. 8. Variation of ratio of time taken by AESA to that by PR–NN procedure with n for different values of d.

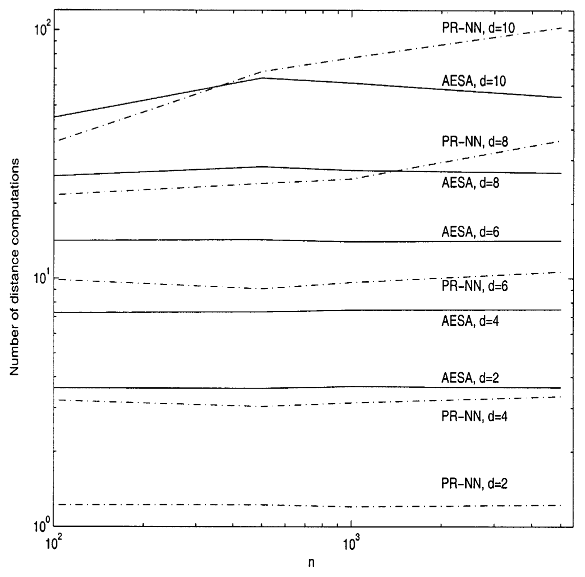


Fig. 9. Variation of number of distance computations with n for different values of d for AESA and PR–NN procedures.

dent of the size of the test set for a given dimensionality of the data. This, in turn, indicates that the time required for computing the nearest neighbor, on an average, will again be independent of the size of the test set. This has also been experimentally verified.

3.2. *Comparison with AESA [8] and LAESA [11]*

The comparative performance of PR–NN with respect to AESA and LAESA is now presented in terms of processing time for computing the nearest neighbor, number of distances to be computed for this purpose, and memory requirements. The size of the training set and the number of dimensions range from 100 to 5000 and 2 to 10, respectively. The size of the test set is kept equal to 1000.

3.2.1. *Processing time for NN computation*

Fig. 8 plots the ratio of the time taken for nearest neighbor computation as required by AESA to that required by PR–NN for different dimensionalities. As can be seen from the figure, this ratio (whose minimum value is > 2) is found to increase with the number of training points (n) as also mostly with the number of dimensions (d). (Note that we were not able to include results for $n = 10,000$ since the time taken by AESA for this n was found to be prohibitively large.)

Note that the processing time reported here for AESA and PR–NN includes the time for the distance computations and the overhead for finding the nearest neighbor, and not the preprocessing time. LAESA is an improvement over AESA in terms of preprocessing (and memory) requirements, and not in terms of distance computations or overhead (computation not strictly applied to distance computation). In fact, the number of distance computations in LAESA is 1.5 times that required in AESA. Therefore, the processing time for LAESA can only be more than that of AESA (or, in other words, will be still poorer compared to PR–NN).

3.2.2. *Number of distance computations*

Fig. 9 shows the performance in terms of the number of distances computed by AESA and PR–NN algorithms. As can be seen from the figure, the number of distances computed by PR–NN procedure is smaller than that of AESA for $d = 2$, 4 and 6. The case is similar even for $d = 8$ and 10, when the sizes of the data set is small. For larger data sets in these dimensionalities, AESA is found to perform better than PR–NN with regard to the number of distances computed. Although for large n and d, the number of distances computed in AESA is less than that for PR–NN, the time taken by the former, as mentioned earlier, is found to be significantly more than that required by the latter (see Fig. 8). This is because of the fact that the processing time required includes the time for distance computations as well as that for the overhead, i.e., computation not strictly applied to distance computation. This overhead is very significant for AESA as compared to PR–NN, thereby making the former computationally intensive. Fig. 10 shows a comparison of AESA and PR–NN with respect to the optimized number of operations involved in the overhead computation of the two methods. (A detailed discussion is provided in the following subsection.) As can be seen, AESA requires significantly more number of operations than PR–NN in overhead computation. This, in

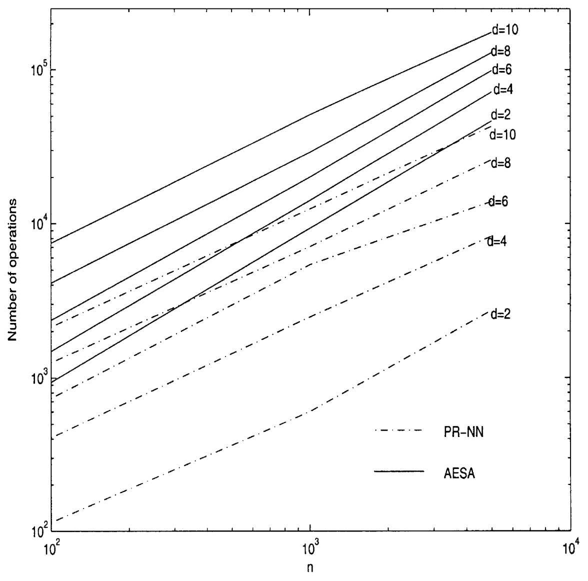


Fig. 10. Variation of number of operations for overhead computation with n for different values of d. The dash–dotted and solid lines correspond to PR–NN and AESA, respectively.

turn, increases the time required by AESA as compared to PR–NN for finding the nearest neighbor. As already mentioned, the time taken by AESA when $n = 10,000$ was prohibitively large. For PR–NN it was obtained in 11.4 s, when the average number of distance computations required was 100.238.

The above discussion, therefore, underlines the utility of the PR–NN procedure with respect to AESA in practical problems. As mentioned in Ref. [11], the number of distance computations in LAESA is about 1.5 times that of AESA. This indicates that in the cases where PR–NN was better than AESA, the former will surely outperform LAESA. On the other hand, in the cases where AESA was better than PR–NN, the difference in performance will be reduced significantly in between PR–NN and LAESA.

3.2.3. Number of optimized operations in overhead computation

In this section, we provide a comparison of AESA and PR–NN in terms of the number of optimized operations involved in the overhead computation i.e., computation not strictly related to distance computation. An indication of the number of distances computed by the two methods has been provided earlier in Fig. 9. Note that the operations include multiplication/division, addition/subtraction and comparison (or, MACs), as is usually considered [13].

For the sake of convenience of the readers, we reproduce here the AESA algorithm [10, pp. 4, 5], and indicate the instructions where we have considered an operation by underlining it.

Algorithm AESA.

```
Input:       Set of prototypes $P$, unknown point $y$, distance
             measure $d$
$D$:         array of precomputed distance between all the
             elements of $P$
Output:      Nearest Neighbor $p^* \in P$ and its distance $d^*$
Variables:   $A : \mathscr{P}(P)$; $p, s, s'$: $P$; $dsy$:real number;
             $G$: array $[P]$ of real numbers
$A = P$
$d^* = \infty$; $p^* = \text{unknown}; G = [0]$;
$s = \text{arbitrary\_selection}(A)$
while $|A| > 0$ do
    $dsy = d(s, y); A = A - \{s\}$
    if $dsy < d^*(\underline{comparison})$ then
      $p^* = s; d^* = dsy$ endif
    $s' = s; s = \text{arbitrary\_selection}(A)$
    $\forall p \in A$ do
      $G[p] = \max(G[p], |D[p, s'] - dsy|)$
        $(\underline{comparison}\ and\ \underline{subtraction})$
      if $G[p] \geqslant d^*$ $(\underline{comparison})$
        then $A = A - \{p\}$
        else if $G[p] < G[s]$ $(comparison)$ then $s = p$ endif
      endif
    end $\forall$
endwhile
end AESA.
```

Similarly, for PR–NN we have considered an operation in the following steps:

A multiplication operator in Step 3. In Step 4 two addition operations are considered for computing $d_y \pm \varepsilon$. Also for each invocation of binary search (for setting the indexes I_l and I_h) we consider two comparison operations, one addition and one division operation. In Step 5 two addition and two comparison operations are considered for computing $y_i \pm \varepsilon$ and checking belongingness within the hypercube for each dimension. Step 6 involves one addition operation corresponding to the increment operator. In Step 7 a comparison operator per element in N'_y is required. Finally, Step 8 requires one comparison operator.

Fig. 10 demonstrates the variation of the number of operations, computed as given above, with the number of prototypes n for different dimensions (mentioned along side the plots). The dash–dotted and solid lines represent the variations for PR–NN and AESA, respectively. It is found from the figure that PR–NN significantly outperforms AESA in terms of the optimized number of operations involved in overhead computation for all the dimensions. It may be mentioned in this context that AESA requires no multiplication operator, while PR–NN does so. However, in real terms, the number of multiplication operators is far less than both the addition and comparison operators. As an illustration, for $n = 5000$ and $d = 10$, the number of additions, comparisons and multiplications in PR–NN is 125.48, 42,908.54 and 61.08, respectively. The number of additions and comparisons, for the same n and d, in case of AESA are 43,889.92

Table 1
Memory requirements for the different schemes. Here n = number of training prototypes, d = number of dimensions and m = number of base prototypes in LAESA [11]

Method	Memory requirement
NN rule	$n \times d$
AESA	$n \times d + n^2$
LAESA	$n \times d + n \times m$
PR–NN	$\geqslant n \times d$ and $\leqslant n \times d + n$

Table 2
Comparative results for the number of distances computed

n	d	Yuncks algorithm	Friedman algorithm	Branch and Bound	PR–NN algorithm
256	2	3.08	18.1	—	1.243
1000	2	4.84	35.7	46	1.203
	4	27.1	239	—	3.151
	6	60.5	481	—	9.63
	8	134	708	—	25.18
	10	493	913	—	77.48
3000	8	248	1851	451	29.725
10,000	10	1127	7251	—	110.238

and 13,1723.78, respectively, which are found to very high as compared to those for PR–NN.

3.2.4. Memory requirement

As mentioned in Ref. [10], AESA has quadratic memory requirements. In LAESA [11], derived from AESA, this was reduced from quadratic to linear in the number of prototypes, at the cost of increase in the number of distance computations. Note that in the PR–NN procedure the memory requirement is linear in the number of prototypes. A comparison of the exhaustive NN scheme, AESA, LAESA and PR–NN in terms of the memory requirement is provided in Table 1.

3.3. Comparison with other methods

As regards the comparison of PR–NN procedure with other existing methods [4–6] other than the two mentioned above, it may be noted that none of these methods reported results with $n = 10,000$. The results are provided in Table 2 (these were taken from already published literature) [6]. As seen from the table, the results of PR–NN procedure are significantly superior for all the values of n and d.

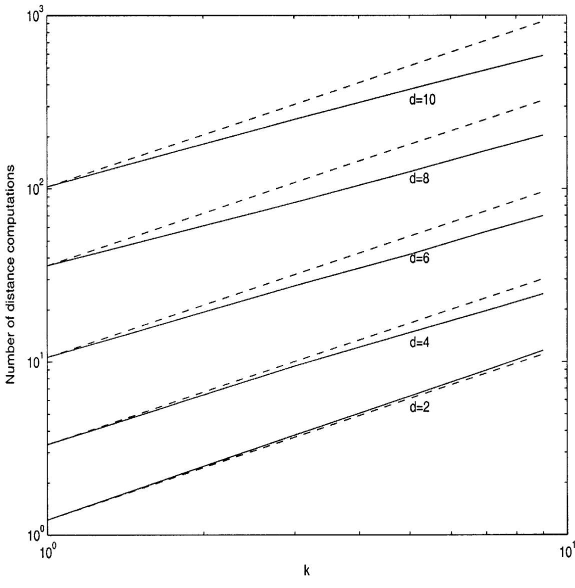


Fig. 11. Variation of number of distance computations with k for different values of d, $n = 5000$. Here the dashed lines indicate a linear scaling by a factor of k.

3.4. Extension to k nearest neighbors

In this subsection the effectiveness of extending the PR–NN procedure to the case of finding the k nearest neighbors is demonstrated experimentally. Fig. 11 shows the variation of the number of distances computed for finding the k nearest neighbors, for $K = 1, 3, 5, 7$ and 9 when 5000 prototypes are considered (i.e., $n = 5000$). As before, we have considered several values d. As can be seen from the figure, the number of distances computed scale linearly with a factor almost equal to k. The dashed lines represent linear scaling by a factor of k. Thus it is found that in practice, the number of distance computations increases by a factor that is smaller than k in all the cases. Interestingly, it is found from Fig. 11 that this factor, in general, decreases with the number of dimensions (d), indicated by the increasing difference between dashed and solid lines for larger d. On the contrary, it may be noted that AESA is specifically designed for finding only the nearest neighbor. Extending it to the case of k nearest neighbors is not straightforward (as with PR–NN).

4. Discussion and conclusions

A very simple preprocessing technique, involving computation and storage of n distances and a reordering of prototypes based on these distances, for computing the nearest neighbor is reported in this article. The modified algorithm retains the intuitive simplicity and flavor of the original NN rule. The new method, PR–NN, is found to require significantly smaller number of distance computations as compared to the exhaustive scheme as well as several other methods available in the literature.

Since the PR–NN procedure involved some overhead not directly related to distance computation, natural concern about its timing requirement with relation to the NN rule arose. It has been experimentally demonstrated that in terms of computation time, the new method is superior to the NN procedure as well as to AESA and LAESA. A comparison of the two methods, in terms of the number of optimized operations required for finding the nearest neighbor also demonstrates the superiority of PR–NN. As mentioned in Ref. [11], the linear overhead of (L)AESA can become a serious bottleneck if the number of prototypes becomes very large.

In this respect it may be noted that the PR–NN procedure may be effectively combined with the concept of triangle inequality used in Refs. [8,10]. This is likely to improve the efficiency of the new method, reducing its computational overhead to a large extent. This line of research is currently being pursued by the authors who are getting encouraging initial results.

Finally, the question of scalability of the PR–NN scheme to the case of computing the k nearest neighbors of a point has been addressed. An investigation for different values of k shows that the algorithm scales linearly with the value of k, with the scaling factor being less than k in practice. Moreover, this factor is found to usually decrease for larger dimensionality of the data.

Acknowledgements

The authors would like to thank the reviewer, whose invaluable comments and suggestions helped immensely in improving the quality of the article and providing some new directions of research.

References

[1] R.O. Duda, P.E. Hart, Pattern Classification and Scene Analysis, Wiley, New York, 1973.

[2] K. Fukunaga, Introduction to Statistical Pattern Recognition, Academic Press, New York, 1990.

[3] J.T. Tou, R.C. Gonzalez, Pattern Recognition Principles, Addison-Wesley, Reading, 1974.

[4] J.H. Friedman, F. Baskett, L.J. Shustek, An algorithm for finding nearest neighbors, IEEE Trans. Comput. C-24 (1975) 1000–1006.

[5] K. Fukunaga, P.M. Narendra, A branch and bound algorithm for computing k-nearest neighbors, IEEE Trans. Comput. C-24 (1975) 750–753.

[6] T.P. Yunck, A technique to identify nearest neighbors, IEEE Trans. Syst. Man Cybern. SMC-6 (1976) 678–683.

[7] I.K. Sethi, A fast algorithm for recognizing nearest neighbors, IEEE Trans. Syst. Man Cybern. C-24 (1975) 1000–1006.

[8] E. Vidal, An algorithm for finding nearest neighbors in (approximately) constant average time, Pattern Recognition Lett. 4 (1986) 145–157.

[9] A. Farago, T. Linder, G. Lugosi, Nearest neighbor search and classification in o(1) time, Problems Control Inform Theory 20 (1991) 383–395.

[10] E. Vidal, New formulation and improvement of the nearest neighbor approximating and eliminating search algorithm, Pattern Recognition Lett. 15 (1994) 1–7.

[11] M.L. Mico, J. Oncina, E. Vidal, A new version of the nearest-neighbor approximating and eliminating search algorithm (asea) with linear preprocessing time and memory requirements, Pattern Recognition Lett. 15 (1994) 9–17.

[12] B. Bhattacharya, D. Kaller, Reference set thinning for the k-nearest neighbor rule, in: Proceedings of the 14th International Conference on Pattern Recognition, Los Alamitos, CA, USA, pp. 238–242, IEEE Computer Society Press, 1998.

[13] V. Ramasubramanian, K.K. Paliwal, Fast nearest-neighbor search algorithms based on approximation-elimination search, Pattern Recognition 33 (2000) 1497–1510.

About the Author—SANGHAMITRA BANDYOPADHYAY did her Bachelors in Physics and Computer Science in 1988 and 1991, respectively. Subsequently, she did her Masters in Computer Science from Indian Institute of Technology (IIT), Kharagpur in 1993 and Ph.D. in Computer Science from Indian Statistical Institute, Calcutta in 1998. Currently she is a faculty member at Indian Statistical Institute, Calcutta, India. Dr. Bandyopadhyay is the first recipient of *Dr. Shanker Dayal Sharma Gold Medal* and *Institute Silver Medal* for being adjudged the best all round post graduate performer in IIT, Kharagpur in 1994. She has worked in Los Alamos National Laboratory, Los Alamos, USA in 1997 as a graduate research assistant, in the University of New South Wales, Sydney, Australia as a post doctoral fellow and in the Department of Computer Science and Engineering, University of Texas at Arlington, USA as a faculty and researcher. Dr. Bandyopadhyay received the Indian National Science Academy (INSA) and the Indian Science Congress Association (ISCA) *Young Scientist Awards* in 2000. Her research interests include Pattern Recognition, Data Mining, Evolutionary Computation, Computer Vision and Parallel & Distributed Systems.

About the Author—UJJWAL MAULIK did his Bachelors in Physics and Computer Science in 1986 and 1989, respectively. Subsequently, he did his Masters and Ph.D. in Computer Science in 1991 and 1997, respectively from Jadavpur University, India. He is currently a faculty in the Department of Computer Science and Technology, Kalyani Government Engineering College, Kalyani University, where he has also small served as the head of the Computer Science Department during 1996–1999. Dr. Maulik has visited Center for Adaptive Systems Application, Los Alamos, New Mexico, USA in 1997 as a scientist, and University of New South Wales, Sydney, Australia as a post-doctoral researcher in 1999. He received the Govt. of India BOYSCAST fellowship in 2001, with which he visited University of Texas at Arlington, USA. His research interests include Parallel and Distributed System, Artificial Intelligence and Combinatorial Optimization, Pattern Recognition, Image Processing and Computer Vision.