Available online at www.sciencedirect.com

SCIENCE DIRECT®

Pattern Recognition Letters 26 (2005) 1554–1567

Pattern Recognition Letters

www.elsevier.com/locate/patrec

A divide-and-conquer approach to the pairwise opposite class-nearest neighbor (POC-NN) algorithm ☆

Thanapant Raicharoen *, Chidchanok Lursinsap

Advanced Virtual and Intelligent Computing Center (AVIC), Department of Mathematics, Faculty of Science, Chulalongkorn University, Bangkok 10330, Thailand

Received 20 June 2004; received in revised form 15 October 2004
Available online 17 February 2005
Communicated by R.P.W. Duin

Abstract

This paper presents a new method based on divide-and-conquer approach to the selection and replacement of a set of prototypes from the training set for the nearest neighbor rule. This method aims at reducing the computational time and the memory space as well as the sensitivity of the order and the noise of the training data. A reduced prototype set contains *Pairwise Opposite Class-Nearest Neighbor* (*POC-NN*) prototypes which are close to the decision boundary and used instead of the training patterns. *POC-NN* prototypes are obtained by recursively iterative separation and analysis of the training data into two regions until each region is correctly grouped and classified. The separability is determined by the *POC-NN* prototypes essential to define the locations of all separating hyperplanes. Our method is fast and order independent. The number of prototypes and the overfitting of the model can be reduced by the user. The experimental results signify the effectiveness of this technique and its performance in both accuracy and prototype rate as well as in training time to those obtained by classical nearest neighbor techniques.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Pattern classification; Nearest neighbor rule; Prototype selection; Prototype replacement

1. Introduction

The *Nearest Neighbor* (*NN*) rule was originally proposed by Cover and Hart (Cover and Hart, 1967; Hart, 1966) in 1966 and has been shown to be very effective in many applications of pattern recognition. One reason for the use of this rule

☆ Supported by the Thailand Research Fund (TRF)(PHD/0173/2545).

* Corresponding author. Tel.: +66 9924 1924; fax: +66 2534 2483.

E-mail addresses: thanapant@avic.sc.chula.ac.th (T. Raicharoen), lchidcha@chula.ac.th (C. Lursinsap).

0167-8655/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2005.01.003

is its conceptual simplicity, which is easy to implement. Moreover, under some continuity assumptions on the underlying distributions, the asymptotic error rate of this rule is at most twice Bayes' probability of errors (Cover and Hart, 1967; Duda et al., 2001). However, it suffers from various drawbacks. Firstly, it requires a large memory space as the entire training data set has to be stored and each test pattern has to be compared with every training pattern. Secondly, it requires a large computational time for finding the neighbors. Thirdly, it is sensitive to the noisy and/or outlier patterns. To alleviate these drawbacks, two approaches, *prototype selection* and *prototype replacement* are introduced (Kuncheva and Bezdek, 1998). Both approaches aim at modifying an original training pattern (prototype) in order to reduce its size as well as to improve classification performance. One of the first and most popularly used techniques of prototype selection is the *Condensing Nearest Neighbor* (*CNN*) proposed by Hart (1968). The main goal of the condensing method is to obtain the reduced and *consistent set* of prototypes (Hart, 1968) to be used with the 1-*NN* rule without error in the training set or in with the *k*-*NN* rule without significantly degrading its performance. The condensing method proceeds by repeatedly selecting the prototypes whenever they cannot be correctly classified by the currently selected set. The whole process is iterated until there are no changes in a complete pass through the initial training set. However, the method does not, in general, yield a minimal size of the consistent subset, and the final size as well as composition of the final condensed set may strongly depend upon the order of presentation data. Since the development of the *CNN*, other methods were proposed successively, such as the *Reduced Nearest Neighbor* (*RNN*) rule proposed by Gates (1972). An algorithm for a *Selective Nearest Neighbor* (*SNN*) decision rule was introduced by Ritter et al. (1975). Tomek presented the *Two Modifications of the CNN* by growing the condensed set using only patterns close to the decision boundary (Tomek, 1976). The way in which pairs of prototypes are selected makes the algorithm very good at preserving the original decision boundaries. Chidananda Gowda and Krishna introduced the concept of *mutual* nearest neighbor neighborhood for selecting patterns close to the decision bounderies (Chidananda Gowda and Krishna, 1979). The position of a prototype in the ordered list of neighbors of its nearest neighbor from an opposite class is used as a way to measure the closeness to boundaries. Several theoretical results on *CNN* have been obtained in (Devroye et al., 1996).

Another well known used technique of prototype selection is the *editing* method (Devijver and Kittler, 1982) proposed by Wilson (1972). The main goal of the editing method is to improve the performance by discarding outliers and possible overlapping among classes rather than prototype reduction. However, the drawbacks of the editing method are that it still leaves too many prototypes in the edited set, and the complexity of computing the edited subset is very high. Therefore, Sanchez proposed the *k*-*Nearest Centroid Neighbors* (*k*-*NCN*) in order to identify and eliminate mislabelled, noisy and atypical training patterns (Sanchez et al., 2003). Several editing experiments are carried out and comparative results are presented in (Ferri et al., 1999). The exploration and exploitation of the synergy among the *NN editing* and *condensing* methods in order to facilitate the use of *NN* techniques in real-world applications was studied in (Dasarathy et al., 2000).

For the prototype replacement approach, one of the first methods, proposed by Chang (1974), repeatedly merges the two nearest neighbors of the same class as long as this merger does not increase the error rate on the training set. One drawback of this method is that it may yield the prototypes that do not characterize the training set well in terms of generalization (Toussaint, 2002).

In order to obstruct the undesirable property of the order dependence of presentation data, several attempts, for example in (Devi and Murty, 2002; Alpaydin, 1997), were suggested to obtain selected prototypes that are less sensitive to this property. However, most of the improvements of *NN* methods cannot avoid overfitting for noisy and/or overlapping data, and do not consider any statistical properties of training data.

This work is focused on developing a new method for obtaining a set of selection prototypes as well as replacement prototypes. Unlike all of these above mentioned methods, our proposed method is based on a divide-and-conquer approach. That is the analogy to partition original training patterns into smaller regions by finding *POC-NN* prototypes for the regions, and then combine *POC-NN* prototypes for the regions into a set of selection prototypes. The rest of this paper is organized into five sections. Sections 2 and 3 present our proposed *POC-NN* method for prototype selection and replacement, respectively, including its analysis. In Section 4, some experiments are described and the performance of our method is evaluated. Section 5 concludes the paper.

2. The methodology of the *POC-NN* for prototype selection

The idea is to isolate a subset of the training set that suffices for nearest neighbor predictions, and throw away the remaining data. Intuitively, it seems important to keep the training points (patterns) that are close to the decision boundaries and on the correct side of those boundaries, while some points far from the boundaries should be discarded. The bottom line of this idea is to find a *POC-NN* pattern for a given two-class training data set and then the remaining patterns can be discarded.

2.1. Finding *POC-NN* pattern algorithm

Let S be a training set of n patterns composed of two subsets $S^{(1)}$ and $S^{(2)}$ whose sizes are $|S^{(1)}| = n^{(1)}$ and $|S^{(2)}| = n^{(2)}$. Both $S^{(1)}$ and $S^{(2)}$ are in different classes, namely classes 1 and 2, respectively, and $S^{(1)} \cap S^{(2)} = \emptyset$. The algorithm to find a *POC-NN* pattern is given as follows.

```
Function FINDING-POC-NN (S: Dataset)

1. Let $S^{(1)}$ and $S^{(2)}$ be two training sets with classes 1 and 2,
   whose sizes are $S^{(1)}$ and $S^{(2)}$, respectively.
2. If $n^{(1)} \geqslant n^{(2)}$
   Then
3.     $\mathbf{x}_m$ = mean of $S^{(1)}$.
3.1.   Let $\mathbf{x}_{p2} \in S^{(2)}$ be the nearest pattern to $\mathbf{x}_m$.
3.2.   Let $\mathbf{x}_{p1} \in S^{(1)}$ be the nearest pattern to $\mathbf{x}_{p2}$.
   Else
4.     $\mathbf{x}_m$ = mean of $S^{(2)}$.
4.1.   Let $\mathbf{x}_{p1} \in S^{(1)}$ be the nearest pattern to $\mathbf{x}_m$.
4.2.   Let $\mathbf{x}_{p2} \in S^{(2)}$ be the nearest pattern to $\mathbf{x}_{p1}$.
   Endif
5. Return $(\mathbf{x}_{p1}, \mathbf{x}_{p2})$ as a POC-NN pattern.
```

Fig. 1(a) shows an example of how finding *POC-NN* algorithm works. There are two classes, 1 and 2. Each pattern in class 1 is denoted by the symbol "+" and each pattern in class 2 is denoted by the symbol "*". The mean of patterns ($\mathbf{x}_m$) in class 1 is denoted by the symbol "◊". The *POC-NN* prototypes $(\mathbf{x}_{p1}, \mathbf{x}_{p2})$ are enclosed in circle symbols "○". This *POC-NN* prototype is performed during the training process and all considered patterns in all classes are considered as the training patterns. The identical *POC-NN* patterns as prototypes are always obtained independent from the different reordering of the training patterns. Moreover, the *POC-NN* patterns have the desirable property similar to the *Support Vectors* (Cortes and Vapnik, 1995; Vapnik, 1998) which induce the optimal separating hyperplane. This is because if all the training vectors are linearly independent, the two closest patterns of opposite classes are support vectors. This proof can be found in (Roobaert, 2000). However, the *POC-NN* algorithm is not guaranteed to find these support vectors. Fig. 1(b) and (c) show the prototypes created by using the *CNN* method (Hart, 1968) which does not guarantee the optimal solution. The condensed prototypes are enclosed in square symbols "□". The obtained prototypes strongly depend on the order of presentation of the training patterns. In this case, it depends on the first pattern of each class. Fig. 1(d) also shows the prototypes by using the *CNN* method which creates the problems of redundant prototypes and also slow convergence.

Once *POC-NN* prototypes are found, a separating linear hyperplane is generated and orthogonally placed in the middle of the distance between these *POC-NN* prototypes. This hyperplane acts as a decision boundary (similar to a *Voronoi Diagram*). The boundary of each region

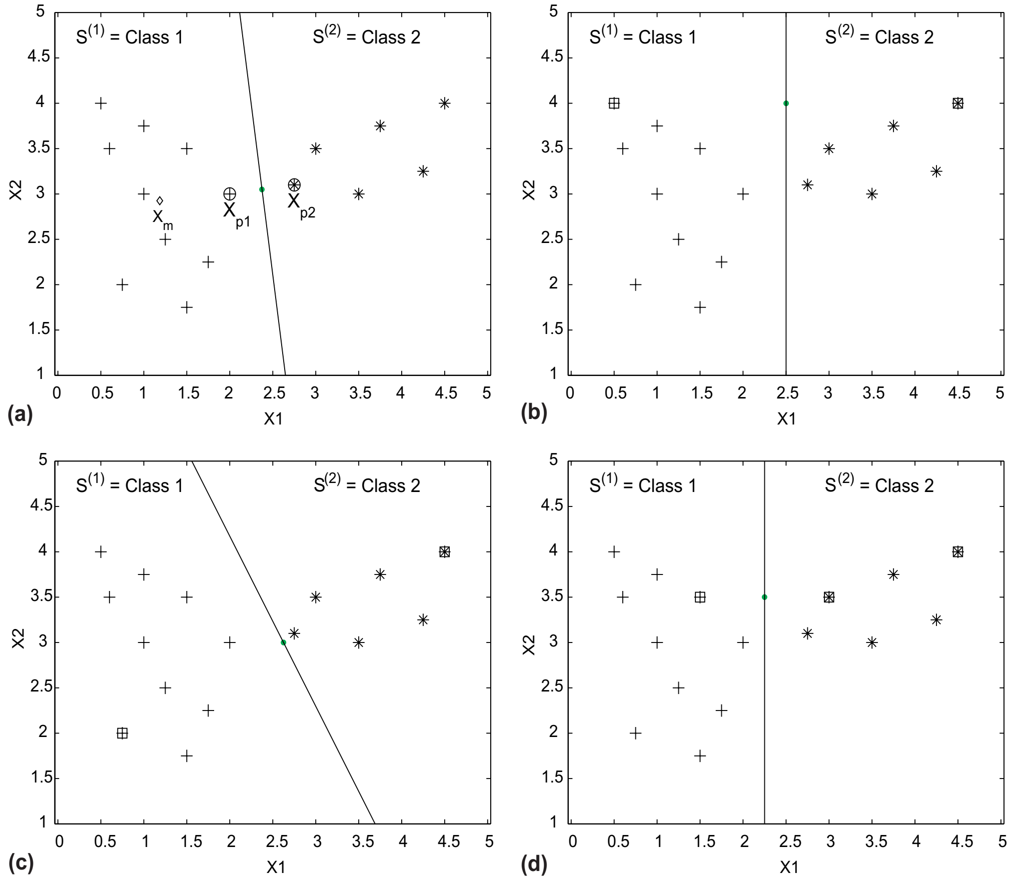


Fig. 1. This example shows how to select the prototypes: (a) our *POC-NN*; (b)–(d) CNN when reordering training patterns.

is defined by the corresponding hyperplane generated by the *POC-NN* prototype lying in each region. The following is the detail of prototype selection by *POC-NN* algorithm.

2.2. Prototype selection by POC-NN algorithm for two-class classification problem

In this section, we present our proposed algorithm for two-class classification problem and in the later subsection, we extend our algorithm for multi-class classification problem.

Prototype selection by POC-NN algorithm for two-class classification

Let S be a training set of n patterns composing of two classes, and *POC-NN-SET* initially be an empty *POC-NN* prototypes set.

Function *SELECTING-POC-NN* (*S: Dataset*)

```
1. Find a POC-NN prototype in $S$ by using
   $(\boldsymbol{x}_{p1}, \boldsymbol{x}_{p2})$ = FINDING-POC-NN ($S$).
2. Determine the center point $\boldsymbol{c} = \frac{\boldsymbol{x}_{p1}+\boldsymbol{x}_{p2}}{2}$.
3. Create a separating hyperplane $H$: $\{\boldsymbol{x} | \boldsymbol{w} \cdot \boldsymbol{x} - b = 0\}$,
   where $\boldsymbol{w} = \frac{\boldsymbol{x}_{p1}-\boldsymbol{x}_{p2}}{\|\boldsymbol{x}_{p1}-\boldsymbol{x}_{p2}\|}$ and $b = \boldsymbol{w} \cdot \boldsymbol{c}$.
4. Save $(\boldsymbol{x}_{p1}, \boldsymbol{x}_{p2})$ and coressponding $H$ into the
   POC-NN-SET.
5. Divide all patterns of $S$ into two regions,
   namely $R1$ and $R2$, where
   $R1 = \{\boldsymbol{x}_i \in S | \boldsymbol{w} \cdot \boldsymbol{x}_i - b \geqslant 0\}$, and
   $R2 = \{\boldsymbol{x}_i \in S | \boldsymbol{w} \cdot \boldsymbol{x}_i - b < 0\}, \forall i, i = 1, \ldots, n$.
6. Find any misclassification in both regions.
7. If any misclassification exists in region $R1$
```

```
     Then
 8.     Consider all data in this region R1 as a new data set
        Call SELECTING-POC-NN (R1).
     Endif
 9.  If any misclassification exists in region R2
     Then
10.     Consider all data in this region R2 as a new data set
        Call SELECTING-POC-NN (R2).
     Endif
11.  If no more misclassification exists
     Then
12.     Return POC-NN-SET as a set of selected prototypes.
        Stop.
     Endif
```

Fig. 2 shows an example of how the algorithm works for non-linearly separable problems. The initial separating line (H_1) created by a *POC-NN* ($\boldsymbol{x}_{p1}, \boldsymbol{x}_{p2}$) is shown in Fig. 2(a). However, this line still creates misclassified training patterns lying on the right side of this line. All the training patterns lying on this side are considered as a new data set. The second line (H_2) as shown in Fig. 2(b) is introduced to resolve the previously misclassified patterns. After performing the algorithm, a set of selecting prototypes corresponding to the *POC-NN-SET* consisting of two pairs of $\{(\boldsymbol{x}_{p1}, \boldsymbol{x}_{p2})\}$ is obtained. The patterns from $S^{(1)}$ are separated into two regions (regions 1 and 2). Each region has a prototype ($\{\boldsymbol{x}_{p1}\}$) representative class labeled 1. The patterns from $S^{(2)}$ are not separated, however, they have two prototypes ($\{\boldsymbol{x}_{p2}\}$) representative class labeled 2.

2.3. Reducing complexity and sensitivity to noise

In order to reduce the complextity and sensitivity to noise as well as to avoid overfitting for overlapping data, we relax the separating condition. A separating hyperplane can be considered as a slab of width alpha (α), called *acceptance interval*. All patterns lying within the acceptance interval of the slab are considered as correctly classified patterns, and also considered as noisy and/or outlier patterns which can be ignored or discarded. Our acceptance interval (α), is defined as follows.

Suppose S be a training data set of n points with two classes and $\boldsymbol{x}_i \in S$.

Definition 1. Let $\{\boldsymbol{x} | \boldsymbol{w} \cdot \boldsymbol{x} - b = 0\}$ be a hyperplane (H). α is an acceptance interval if $|\boldsymbol{w} \cdot \boldsymbol{x}_i - b| < \alpha$, and $\alpha > 0$ and all $\boldsymbol{x}_i$ are correctly classified.

The acceptance interval is proportional to the distance d between two *POC-NN* prototypes, and is defined priori to the training process. In the other word, α is defined as $\alpha_r \times d$ for $0 < \alpha_r < 1$. This α_r is called α-ratio.

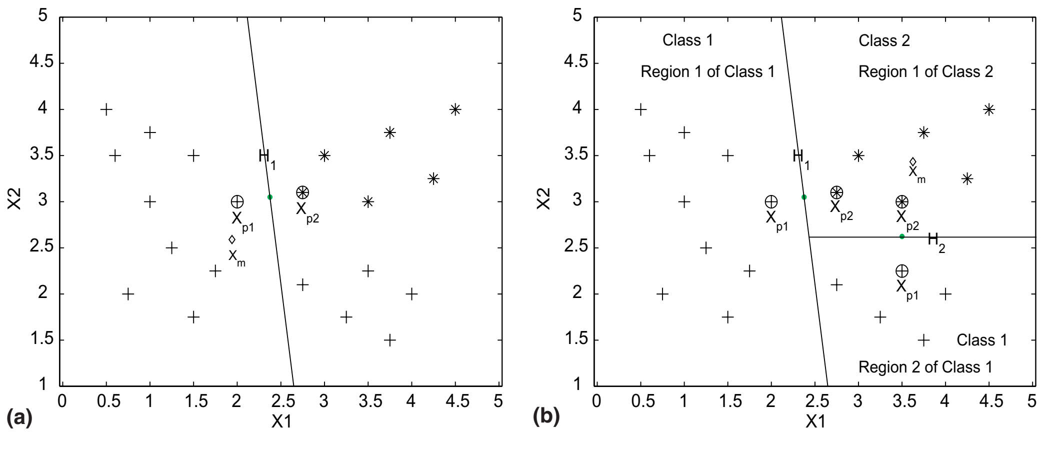


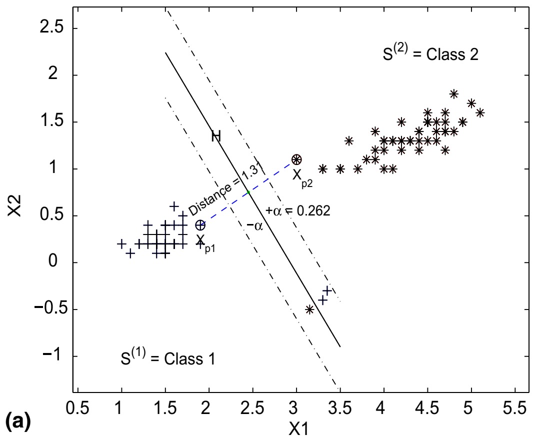
Fig. 2. This example shows how the proposed algorithms works: (a) initial separating line (H_1); (b) second separating line (H_2).

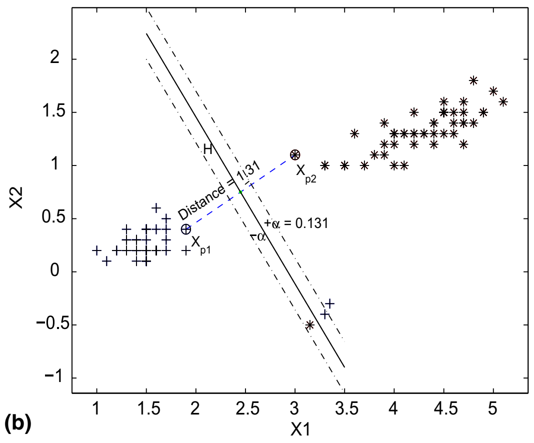
These α values for each hyperplane are not necessary the same values, but are dependent on α-ratio and the distance between corresponding *POC-NN* prototypes.

Fig. 3 shows an example of an acceptance interval defined by giving the α-ratio 1:5 or 0.2 (a), and 1:10 or 0.1 (b) and (c). In Fig. 3(a), our algorithm will stop after one iteration and generate two *POC-NN* prototypes. Each *POC-NN* is a prototype representative for each class, even though there are three missclassified patterns lying within the acceptance interval. They are considered as noisy and/or outlier patterns which are ignored. In Fig. 3(b), there are only two instead of three missclassified patterns lying within the acceptance interval and considered as noisy and/or outlier patterns. One missclassified pattern is still there, therefore, the algorithm will continue and generate the other two *POC-NN* prototypes shown in Fig. 3(c).

2.4. Analysis of the selecting POC-NN algorithm

There are some interesting consequences in selecting *POC-NN* prototype worth discussing. Given a training set S of n patterns ($n > 1$) in d dimensions composing of two subsets $S^{(1)}$ and $S^{(2)}$, whose sizes are $n^{(1)}$ and $n^{(2)}$. Both $S^{(1)}$ and $S^{(2)}$ are in different classes, namely classes 1 and 2, respectively, and there never exist two





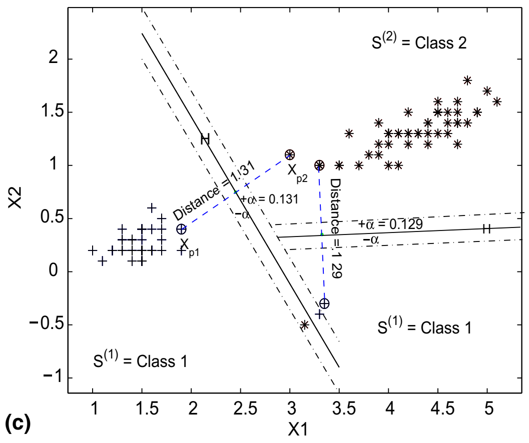


Fig. 3. All points lying inside of the slab with radius alpha (α) are considered as noisy patterns. The ratio between an α value and the distance between corresponding *POC-NN* prototypes is pre-defined by a user. (a) α-Ratio is equal to 1:5(0.2). (b,c) α-Ratio are equal to 1:10(0.1).

patterns with different class label on the position ($S^{(1)} \cap S^{(2)} = \emptyset$).

Property 1. The selecting *POC-NN* algorithm converges after m iterations, where m is the number of *POC-NN* patterns having the values between 1 and $n - 1$.

Proof. We start our proof with $n^{(1)}$ and $n^{(2)} = 1$, so these two patterns are *POC-NN* pattern to each other and its hyperplane can be constructed to make them linearly separable. If we add a new pattern into this space, we always have two cases:

Case 1: A new pattern lies on the correct side, so all data are still linearly separable and correctly classified. The algorithm stops after one iteration.

Case 2: A new pattern lies on the wrong side. Here, we need to consider only the data lying on the wrong side. A new pattern will become a new *POC-NN* pattern and a new hyperplane that makes these data linearly separable, is introduced. The algorithm will stop after two iterations.

If we gradually add a new pattern until there are n patterns of training set S, we still always have two cases as before. The algorithm will stop after $n - 1$ iterations.

Assume that the algorithm does not converge. This would only be possible if no hyperplane could be constructed. But note that the hyperplane cannot be constructed if the nearest patterns of the two classes in data group cannot be found. It would only be possible if the data group has only the same class data and that is also the stop condition of our algorithm. This is a contradiction and hence non-convergence does not occur. $\square$

The consequence of this property yields that all training patterns are separated into the regions of correctly classified classes after convergence. Therefore, a set of patterns in each region is a *consistent set*. However, the whole set of the selected *POC-NN* prototypes may not be a *consistent set*. This proof can be shown by giving a counterexample that the obtained set of prototypes generated by our algorithm does not give 100% accuracy on the training set by using 1-*NN* rule. Similar to the Tomek's algorithm, the consistency is not guaranteed. A counterexample has been shown in (Toussaint, 1994). However, there is no theoretical evidence on how the consistency of the condensed set relates to the generalization abilities. It may have an arbitrarily poor performance when applied to unseen patterns as shown by some experimental examples in Section 4.

Property 2. The time complexity of the selecting *POC-NN* algorithm is $O(dn^2m)$.

Proof. The upper bound of the time complexity is derived step by step in the following analysis. In step 1 of the selecting *POC-NN* algorithm, the calculation for finding a *POC-NN* prototype takes in worst case $O(dn^2)$, and in steps 5 and 6, the calculation takes $O(dn)$. The time complexity is bounded by $O(dn^2 + dn)$ after one iteration. If the algorithm does not converge, step 1 of the algorithm will continue with at most in worst case $(n - 1)$ patterns. The time complexity is bounded by $O(d(n - 1)^2 + d(n - 1))$ after two iterations. From the property 1, our algorithm converges after m iterations, therefore, the total time complexity after m iterations is bounded by $O(dn^2 + dn) + O(d(n - 1)^2 + d(n - 1)) + \cdots + O(d1^2 + d1)$ which is equal to $O(dn^2m)$. $\square$

Comparison with *Condensing* and *Editing NN* algorithm, the complexity of computing the condensed subset and edited subset are $O(dn^3)$ and $O(d^3 n^{[d/2]} \ln n)$, respectively (Toussaint, 2002; Duda et al., 2001). Both time complexities are higher than the selecting *POC-NN's*. In addition, the comparison with the other two modified condensing methods which attempt to keep only prototypes close to the decision boundary, Tomek's algorithm (Tomek, 1976) and Gowda–Krishna's algorithm (Chidananda Gowda and Krishna, 1979), the computational complexity of both algorithms is considerable higher than the original *CNN* algorithm. In case of Tomek's algorithm, only the step for finding a special subset, called *Gabriel Neighbors* (Toussaint, 1994), already takes $O(n^3)$ time. For a large number n such as that in OCR applications shown by USPS data (USPS, 1994) experimental example in Section 4, it is not feasible. Gowda–Krishna's algorithm takes more time than Tomek's algorithm in some cases as shown in (Chidananda Gowda and Krishna, 1979), since it requires a preprocess for finding the *Mutual Neighborhood Value* (*MNV*) of each pattern to order the patterns according to *MNV*, and a post-process for reducing the number of condensed set of prototypes.

Property 3. The selecting *POC-NN* algorithm is order independent.

Proof. The training set can have $n!$ permutations order of how to present the data. In steps 3 and 4 of the finding a *POC-NN* pattern algorithm, the centroid or mean of a set of points (patterns) is calculated. The addition remains invariant to the order of the data elements. Thus, the mean operation is order independent. Finding the nearest point to the centroid is also order independent, since the centroid and the distances between the centroid and all points from different classes are fixed values. The shortest distance is found and the corresponding point is selected as the representative prototype. If there is a tie between two or more points for the closest distance from the centroid, the point with the smallest value of the mean of its feature is selected. If this is the same, the point with the smaller value of feature 1 is selected. If this is still the same, the point with the smaller value of feature 2 is selected etc. So the distance measure is order independent and the minimum of these distances remains the same no matter what the order of the presentation data is. The other steps in the algorithm are obviously order independent, thus the selecting *POC-NN* algorithm is order independent. □

2.5. Algorithm for multi-class classification problem

In this section, we extend our proposed algorithm for multi-class classification problem by using a combination of many two-class classifiers into a multi-class classifier. In contrast to the standard approach to the k-class problem by a '*one-against-rest*' $(1-v-r)$ like scheme, we use the alternative approach, so called '*one-against-one*' $(1-v-1)$ scheme using pairwise classification with $\frac{k(k-1)}{2}$ binary classifiers.

We classify all k-class training data into pairwise two-class training sets and perform our selecting *POC-NN* algorithm on each two-class training set. It is possible that the same pattern becomes a *POC-NN* prototype more than once, when it has been selected by another combination of pairwise two-class training set. However, it is still the same representative prototype for its class label. After performing our algorithm for each two-class pair, we obtain *POC-NN* prototypes representative for each class. The time complexity for multi-class classification will be increased to $\mathrm{O}\left(\frac{k(k-1)}{2}(dn^2m)\right)$, where n is the number of patterns with two classes and m is the number of *POC-NN* prototypes.

3. The methodology of *POC-NN* for prototype replacement

After performing our prototype selection by *POC-NN* algorithm on a training set, this training set is separated into many regions of correctly classified classes. Each region has representative *POC-NN* prototypes being elements of a reference set. In this section, the original training data set are replaced by a number of prototypes that do not necessarily coincide with any pattern in training set. Our *POC-NN* method for prototype replacement is applied by replacing all *POC-NN* prototypes in each region with the mean of patterns in its region. Since, in the statistical principle, the mean of a data set is the bestunderstood measure of central tendency for a quantitative data set (McClave et al., 2001), therefore the mean can be a good representative pattern for all patterns in its region. In consequence, the number of replacement prototypes in general will be decreased when compared with the number of selection prototypes.

3.1. Prototype replacement by POC-NN algorithm for two-class classification problem

We propose our prototype replacement by *POC-NN* algorithm by modification of the prototype selection by *POC-NN* algorithm for Two-Class Classification as follows.

Let *MOR-NN-SET* initially be an empty set of mean of region prototypes.

```
Function REPLACING-POC-NN (S: Dataset)

1.  Find a POC-NN prototype in S by using
    $(x_{p1}, x_{p2}) = FINDING\text{-}POC\text{-}NN(S)$.
2.  Determine the center point $c = \frac{x_{p1}+x_{p2}}{2}$.
3.  Create a separating hyperplane H:
    $\{x | w \cdot x - b = 0\}$,
    where $w = \frac{x_{p1}-x_{p2}}{\|x_{p1}-x_{p2}\|}$ and $b = w \cdot c$.
4.  Save $(x_{p1}, x_{p2})$ and coressponding H into the POC-NN-SET.
5.  Divide all patterns of S into two regions, namely R1 and R2, where
    $R1 = \{x_i \in S | w \cdot x_i - b \geqslant 0\}$, and
    $R2 = \{x_i \in S | w \cdot x_i - b < 0\}, \forall i, i = 1, \ldots, n$.
6.  Find any misclassification in both regions.
7.  If any misclassification exists in region R1
    Then
8.      Consider all data in this region R1 as a new data set
        call REPLACING-POC-NN (R1).
    Else
9.      Save $x_{mor}$ being the mean of R1 into the MOR-NN-SET.
    Endif
10. If any misclassification exists in region R2
    Then
11.     Consider all data in this region R2 as a new data set
        call REPLACING-POC-NN (R2).
    Else
12.     Save $x_{mor}$ being the mean of R2 into the MOR-NN-SET.
    Endif
13. If no more misclassification exists
    Then
14.     Save $x_{mor}$ being the mean of R1 into the MOR-NN-SET, and
        Save $x_{mor}$ being the mean of R2 into the MOR-NN-SET.
15.     Return MOR-NN-SET as a set of replacement prototypes.
        Stop.
    Endif
```

Fig. 4 shows an example of how the prototype replacement algorithm works on the same data set shown in Fig. 2. After the algorithm converges, the training set is separated into three regions of correctly classified classes. Therefore, we obtain three *MOR-NN* prototypes (x_{mor}) used to replace the previous four *POC-NN* prototypes shown in Fig. 2(b). Normally, the number of *MOR-NN* prototypes is less than the number of *POC-NN* prototypes.

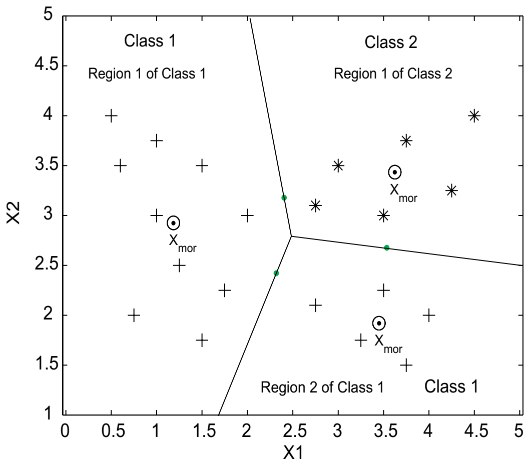


Fig. 4. This example shows MOR-NN prototypes indicated by the symbol "⊙".

3.2. Analysis of the replacing POC-NN algorithm

Property 4. The replacing POC-NN algorithm is order independent.

Proof. As shown in Property 3, the selecting *POC-NN* algorithm is order independent. The replacing *POC-NN* algorithm is almost identical except that it additionally calculates the mean of each region. Since the mean operation is order independent, our replacing POC-NN algorithm is also order independent. □

4. Experimental results

Our algorithms are tested and evaluated on a number of standard classification datasets of benchmarks, both artificial and real. These data sets are taken from UCI Repository of machine learning databases (Murphy and Aha, 1994)

Table 1
Properties of the data sets used

Datasets	No. of features	No. of classes	No. of trg. patterns	No. of test patterns
1. USPS	256	10	7291	2007
2. DNA	180	3	2000	1186
3. Sonar	60	2	104	104
4. Vowel	10	11	528	462
5. Wine	9	7	118	60
6. Liver	6	2	230	115
7. Thyroid	5	3	143	72
8. Iris	4	3	100	50
9. Spiral	2	2	129	65

except for the first and second ones which are the USPS data (USPS, 1994) and DNA from Statlog (Michie et al., 1994). The properties of the datasets are given in Table 1. In the first four data sets, the training and test sets are separated. In the others, we separated the original data sets into training and test sets (2:1 ratio) by using the following criteria. The first pattern belongs to the test set, and the second and third patterns belong to the training set. The fourth pattern again belongs to the test set, and the fifth and sixth patterns belong to the training set, and so on. We use Euclidean distance (L_2-norm) to measure the dissimilarity and $1 - NN$ rule to test the results in all experiments.

The results obtained on these datasets are reported in Table 2. The prototype rate (PR%) computed from the percentage of prototypes to all training patterns, and the accuracy rate (AR%) computed from the percentage of correctly predicted test patterns to all test patterns obtained using the *selecting* (*S*-) and *replacing* (*R*-) *POC-NN*, the *Condensed Nearest Neighbor* (*CNN*), the *Tomek's algorithm* (*Tomek*), the *Gowda–Krishna's algorithm* (*GKA*), and the accuracy rate (AR%) obtained using all prototypes (*NN*) are given for each data set. The number of prototypes corresponding to the PR% obtained using the *R-POC-NN* algorithm normally is less than the number of prototypes obtained from using the *S-POC-NN* algorithm. N/A (for not applicable/available) in the Table 2 signifies that the learning scheme did not finish training. If learning could not be completed within the time period of two weeks then it was terminated and marked N/A.

Our results are mostly better than the results obtained by the *CNN*, especially, the accuracy rates using *POC-NN* are better than *CNN* in all cases, and better than *Tomek* and *GKA* in all cases except Liver and Thyroid. However, the prototype rates using *POC-NN* are better than the other algorithms in some cases. In the cases of wine, thyroid and iris data, the selecting *POC-NN* shows better results in both accuracy and prototype rates than the results from the *CNN*. In the cases of sonar, thyroid and spiral data, the replacing *POC-NN* also shows better results in both accuracy and prototype rates than results from the *CNN*. Moreover, in case of vowel and thyroid

Table 2
The comparison results of *S-POC-NN*, *R-POC-NN*, *CNN*, *Tomek*, *GKD*, and *NN*

Datasets	*S-POC-NN*		*R-POC-NN*		*CNN*		*Tomek*		*GKA*		*NN*	
	PR%	AR%	PR%	AR%	PR%	AR%	PR%	AR%	PR%	AR%	PR%	AR%
1. USPS	28.20	93.42**	29.79	**93.57*** **	11.84	91.58	N/A	N/A	N/A	N/A	100.00	94.37
2. DNA	60.70	74.37	58.65	**87.77*** **	44.30	72.30	N/A	N/A	N/A	N/A	100.00	76.39
3. Sonar	60.58	80.77	51.92	**90.38**	62.50	**90.38**	56.73	88.46	50.00	87.50	100.00	91.34
4. Vowel	50.19	54.11**	56.82	**57.14*** **	19.69	47.19	20.83	51.52	18.56	55.84	100.00	56.28
5. Wine	43.22	**71.67**	38.13	58.33	44.07	66.67	42.37	68.33	38.98	68.33	100.00	68.33
6. Liver	58.69	54.78	43.47	53.04	56.52	53.91	55.65	**58.26**	50.00	57.39	100.00	59.91
7. Thyroid	20.27	88.89	15.38	91.67	20.97	88.89	16.08	**95.83**	15.38	90.28	100.00	91.67
8. Iris	17.00	**98.00**	16.00	96.00	20.00	**98.00**	18.00	**98.00**	16.00	**98.00**	100.00	98.00
9. Spiral	56.59	**95.38***	35.66	92.31	37.98	89.23	37.98	93.85	34.11	89.23	100.00	96.92

Best accuracy rates (AR%) among these algorithms except *NN* are bold-faced. Tomek and GKA are not applicable (N/A) on USPS and DNA, because they require unacceptable time to run on these data. The symbols "***", "**" and "*" indicate the statistically significant difference to *CNN* at the level 99%, 95% and 90%, respectively.

Table 3
The results comparisons of *POC-NN* and the other algorithms by conducting the three-fold ($K = 3$) and five-fold ($K = 5$) cross-validation

K-Fold	Datasets	*S-POC-NN*		*R-POC-NN*		*CNN*		*Tomek*		*GKA*	
		PR%	AR%	PR%	AR%	PR%	AR%	PR%	AR%	PR%	AR%
$K = 3$	Wine	43.82	**73.05**	37.67	58.43	41.86	71.37	40.45	**73.05**	38.20	72.50
	Liver	57.35	**60.64****	44.83	57.76	59.81	58.07	57.78	57.19	50.06	57.46
	Thyroid	18.15	90.25	14.42	88.38	19.53	90.24	16.05	**93.49**	14.42	90.23
	Iris	20.67	96.00	17.67	**98.67**	16.00	94.00	14.67	98.00	13.00	92.00
	Spiral	58.50	**96.39**	37.88	91.75	38.40	92.79	38.14	93.82	35.56	91.78
$K = 5$	Wine	42.42	71.86	37.08	64.60	40.45	**73.59**	37.93	73.57	36.52	73.57
	Liver	58.91	**60.29****	46.74	55.65	58.70	59.42	58.26	59.13	53.12	60.00
	Thyroid	16.05	89.77	13.02	91.63*	16.86	89.77	16.05	**94.42**	14.19	93.49
	Iris	8.42	**99.00***	7.17	98.67	7.50	96.33	13.83	98.67	12.00	94.00
	Spiral	60.17	**97.41**	36.46	90.74	33.89	96.90	34.79	**97.41**	34.67	94.35

Best accuracy rates (AR%) are bold-faced.

Table 4
The results comparisons of *S-POC-NN* by using without α-ratio ($\alpha_r = 0$) and with different-ratio ($\alpha_r <> 0$), and using *CNN*

Datasets	*S-POC-NN* ($\alpha_r = 0$)			*S-POC-NN* ($\alpha_r <> 0$)				*CNN*		
	Prototypes		AR%	α_r ratio	Prototypes		AR%	Prototypes		AR%
	#	%			#	%		#	%	
1. USPS	2056	28.20	**93.42****	0.25	631	**8.66**	89.16	863	11.84	91.58
2. DNA	1214	60.70	**74.37***	0.05	793	**39.65**	73.19	886	44.30	72.30
3. Sonar	63	60.58	80.77	0.05	58	**55.77**	80.77	65	62.50	**90.38**
4. Vowel	265	50.19	**54.11****	0.8	103	**19.51**	50.22	104	19.69	47.19
5. Wine	51	43.22	71.67	0.3	49	**41.53**	**71.67**	52	44.07	66.67
6. Liver	135	58.69	54.78	0.5	95	**41.30**	**54.78**	130	56.52	53.91
7. Thyroid	29	20.27	88.89	0.25	22	**15.38**	**88.89**	30	20.97	**88.89**
8. Iris	17	17.00	98.00	0.5	14	**14.00**	**98.00**	20	20.00	**98.00**
9. Spiral	73	56.59	95.38*	0.1	72	55.81	**95.38***	49	**37.98**	89.23

Best accuracy and prototype rates between *S-POC-NN* and *CNN* are bold-faced. The symbol "**" and "*" indicate the statistically significant difference to *CNN* at the level 95% and 90%, respectively.

Table 5
The time comparisons of *POC-NN*, and the others in second

Datasets	Training time is second				
	S-POC-NN	*R-POC-NN*	*CNN*	*Tomek*	*GKA*
1. USPS	297,650	298,700	560,370	N/A	N/A
2. DNA	109,277	11,048	74,474	N/A	N/A
3. Sonar	8.94	9.03	90.20	925.32	882.68
4. Vowel	55.18	55.78	81.29	14,045.00	2873.77
5. Wine	3.29	3.32	20.15	101.51	189.63
6. Liver	4.49	4.57	38.07	620.41	1021.83
7. Thyroid	1.38	1.42	6.65	56.54	24.89
8. Iris	0.80	0.81	2.56	30.42	8.87
9. Spiral	0.66	0.69	8.77	41.81	33.12

Tomek and GKA algorithm are not applicable (N/A) of USPS and DNA, since they require unacceptable time (more than two weeks) to run on these data set.

data, the replacing *POC-NN* even gives better accuracy rate than *NN* by using all the training set as prototypes. So the higher accuracy rate of the training data cannot guarantee to obtain a better accuracy rate of test (untrained) data. The consistent set is not necessary related to the generalization abilities.

From these results, we formulated the hypothesis that using the *POC-NN* algorithm is more effective than using *CNN* algorithm. To determine if this hypothesis was statistically significant, we analyzed the results using a test of significance involving differences of proportions (Hayter, 2002). In the Table 2, this test shows that the level of significance of the hypothesis is 0.01, 0.05 and 0.1 indicated by symbols “***”, “**” and “*”, which provide a 99%, 95% and 90%, respectively, confidence level according to one-tailed proportions of the normal curve that it is correct.

To increase statistical significance of the results on the data set whose training and test sets are not separated, we conduct the *K*-fold cross-validation technique which probably is the simplest and most widely used method for estimating prediction error

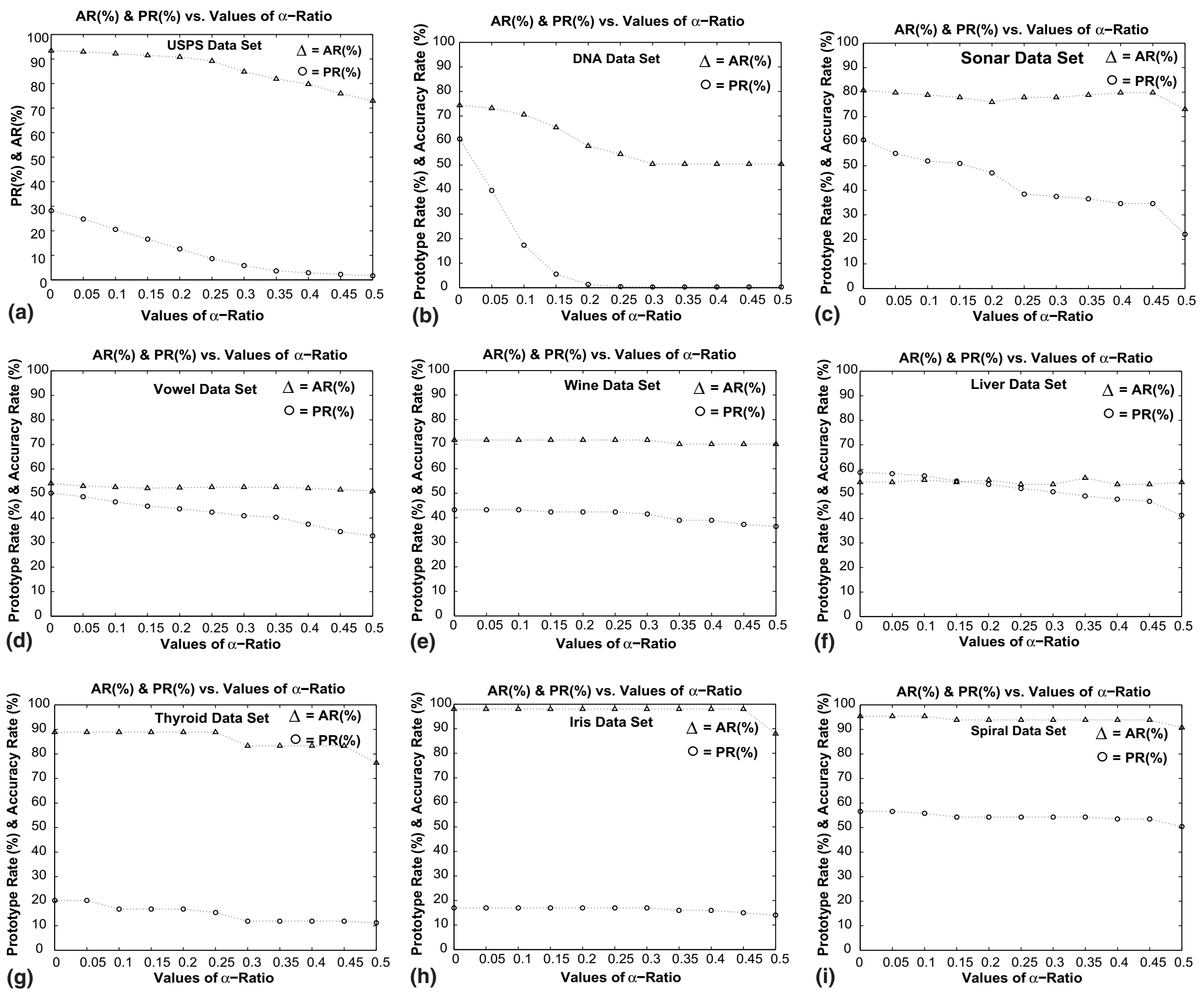


Fig. 5. Prototype rate (PR%) and accuracy rare (AR%) as a function of α-ratio: (a) USPS, (b) DNA, (c) sonar, (d) vowel, (e) wine, (f) liver, (g) thyroid, (h) iris and (i) spiral.

(Hastie et al., 2001), and report the average cross-validation estimate of prototype and accuracy rate as shown in Table 3. Comparison with the results in Table 2, for the three-fold cross-validation ($K = 3$), the *S-POC-NN* still shows better results in accuracy rate than results from the *CNN* in all cases, and shows the best results in accuracy rate in three cases. For the five-fold cross-validation ($K = 5$), the *S-POC-NN* still shows better results in accuracy rate than the others in all cases except the Wine. The symbols "**" and "*" also indicate 95% and 90%, respectively, confidence interval for estimating the difference between accuracies of *S-POC-NN*, *R-POC-NN* and *CNN* using a one-tailed paired *t*-test (Mitchell, 1997).

In order to reduce the number of prototypes, the concept of *acceptance interval* (α) is considered. Table 4 shows the results performed on the same datasets by using the *S-POC-NN* with different α-ratio and *CNN*. Most accuracy rates remain the same by reducing the number of prototypes. In many cases, the *S-POC-NN* with α-ratio gives better both accuracy and prototype rates than the *CNN*. In the case of DNA, sonar and vowel, there is a significant reduction in the number of prototypes used. So the reduced set of prototypes is very useful. By using the concept of *acceptance interval* (α), we can regulate the value of α in order to control the complexity and avoid the overfitting of our model. Fig. 5 shows that the number of *POC-NN* prototypes and also the prototype rate that depend on choosing a prior value of an α-ratio. The α-ratio can also be employed to define the prototype and accuracy rates. In most cases, the accuracy and prototype rates decrease when the α-ratio increases. However, in cases of Sonar and Liver, we obtain the better accuracy rate with lesser prototype rate, even though the α-ratio is increased.

We also compare the training and testing time required for the results shown in Table 2. The less require computational time the more efficiency for an algorithm, even though it is carried out offline. The time comparisons are summerized in Table 5. All experiments were done on the same Pentium III-1GHz computer with 256 MB RAM, and all algorithms were implemented in MatLab version 6.5.We implemented these five different training algorithms in different training programs used to obtain the prototype set in each data set. The selecting *POC-NN* algorithm has the best training time for all datasets. In case of sonar, the *POC-NN*'s training time is approximately 10 times faster than the *CNN*'s training time, and approximately 100 times faster than the *Tomek*'s and the *GKA*'s training time. The *Tomek* and the *GKA* algorithms require very high the computational time and are not applicable for some real world problems, such as USPS and DNA. The time and space complexity of our *POC-NN* method are very competitive.

5. Conclusion

A new *POC-NN* method based on divide-and-conquer approach to the selection and replacement of a set of prototypes has been proposed. For a finite number of training data, our algorithm always converges with all patterns separated into the regions of correctly classified classes. The time complexity required to execute our algorithm is in the order of $O(dn^2m)$. Our method is fast as well as simple, and moreover, it can overcome the undesirable property of the order dependence and the sensitivity of noisy data. The prototype and accuracy rates can also be regulated by a user-defined parameter, called an acceptance interval ratio (α_r). The prototype and accuracy rates obtained have been compared with *CNN*, *Tomek*, *GKA*, and *NN* with all the training patterns. For most of the results, our method showed better performances in both prototype and accuracy rates than the results from the others. For all the above mentioned results, our method showed the best training time. The relationship between the α_r and the average distance among data as well as its variance are not studied here. The value of α_r should be locally adaptive, according to the distribution nature of the data. This issue is essential and worth for further investigation.

References

Alpaydin, E., 1997. Voting over multiple condensed nearest neighbors. Artificial Intell. Rev. 11, 115–132.

Chang, C.L., 1974. Finding prototypes for nearest neighbor classifiers. IEEE Trans. Computer 23 (11), 1179–1184.

Chidananda Gowda, K., Krishna, G., 1979. The condensed nearest neighbor rule using the concept of mutual nearest neighborhood. IEEE Trans. Inform. Theory 25 (4), 488–490.

Cortes, C., Vapnik, V.N., 1995. Support Vector Networks. Mach. Learn. 20 (3), 273–297.

Cover, T.M., Hart, P.E., 1967. Nearest neighbor pattern classification. IEEE Trans. Inform. Theory 13, 21–27.

Dasarathy, B.V., Sanchez, J.S., Townsend, S., 2000. Nearest neighbor editing and condensing tools-synergy exploitation. Pattern Anal. Appl. 3, 19–30.

Devi, V.S., Murty, M.N., 2002. An incremental prototype set building technique. Pattern Recognit. 35, 505–513.

Devijver, P.A., Kittler, J., 1982. Pattern Recognition: A Statistical Approach. Prentice-Hall, Englewood Cliffs, NJ.

Devroye, L., Gyorfi, L., Lugosi, G., 1996. A Probabilistic Theory of Pattern Recognition. Springer-Verlag, Inc, New York.

Duda, R.O., Hart, P.E., Stock, D.G., 2001. Pattern Classification. John Wiley and Sons, Inc.

Ferri, F.J., Albert, J.V., Vidal, E., 1999. Considerations about sample-size sensitivity of a family of edited nearest neighbor rules. IEEE Trans. Systems Man Cybernet. B 29 (5), 667–672.

Gates, G.W., 1972. The reduced nearest neighbor rule. IEEE Trans. Inform. Theory 18, 431–433.

Hart, P.E., 1966. An asymptotic analysis of the nearest-neighbor decision rule. Stanford Electron. Lab., Stanford, Calif., Tech. Rep., SEL-66-016, 1828-2.

Hart, P.E., 1968. The condensed nearest neighbor rule. IEEE Trans. Inform. Theory 14 (3), 515–516.

Hastie, T., Tibshirani, R., Friedman, J., 2001. The Elements of Statistical Learning. Springer Series in Statistics, Springer-Verlag.

Hayter, A.J., 2002. Probability and Statistics for Engineers and Scientists. Duxbury.

Kuncheva, L.I., Bezdek, J.C., 1998. Nearest prototype classification: Clustering, genetic algorithm, or random search? IEEE Trans. Systems Man Cybernet. C 28 (1), 160–164.

McClave, J.T., Benson, P.G., Sincich, T., 2001. Statistics for Business and Economics. Prentice Hall.

Michie, D., Spiegelhalter, D.J., Taylor, C.C., 1994. Machine learning, neural and statistical classification. Available from <ftp://ftp.ncc.up.pt/pub/statlog>.

Mitchell, T., 1997. Machine Learning. McGraw Hill.

Murphy, P.M., Aha, D.W., 1994. UCI Repository of machine learning databases [http://www.ics.uci.edu/mlearn/MLRepository.html], Department of Information and Computer Science, University of California, Irvine, CA.

Ritter, G.L., Woodruff, H.B., Lowry, S.R., Isenhour, T.L., 1975. An algorithm for a selective nearest neighbor rule. IEEE Trans. Inform. Theory 23, 1179–1184.

Roobaert, D., 2000. DirectSVM: A simple support vector machine perceptron. In: Proceedings of IEEE International Workshop on Neural Networks for Signal Processing, Sydney, Australia.

Sanchez, J.S., Barandela, R., Marques, A.I., Alejo, R., Badenas, J., 2003. Analysis of new techniques to obtain quality training sets. Pattern Recognit. Lett. 24, 1015–1022.

Tomek, I., 1976. Two modifications of CNN. IEEE Trans. Systems Man Cybernet. Cybernet. 2, 769–772.

Toussaint, G.T., 1994. A counterexample to Tomek's consistency theorem for a condensed nearest neighbor. Pattern Recognit. Lett. 15, 797–801.

Toussaint, G., 2002. Proximity Graphs for Nearest Neighbor Decision Rules: Recent Progress. In: Proceedings of 34 th Symposium on Computing and Statistics, Montreal, Canada.

USPS, 1994. Machine learning, neural and statistical classification. Available from <fpt://ftp.kyb.tuebingen.mpg.de/pub/bs/data/>.

Vapnik, V.N., 1998. Statistical Learning Theory. Wiley, New York.

Wilson, D.L., 1972. Asymtotic properties of nearest neighbor rules using edited data. IEEE Trans. Systems Man Cybernet. Cybernet. 2, 408–420.