PERGAMON

Pattern Recognition 35 (2002) 505–513

PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY

www.elsevier.com/locate/patcog

An incremental prototype set building technique

V. Susheela Devi[a], M. Narasimha Murty[b,*]

[a]Department of Electrical Engineering, Indian Institute of Science, Bangalore 560 012, India
[b]Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560 012, India

Received 27 April 2000; received in revised form 30 November 2000

Abstract

This paper deals with the task of finding a set of prototypes from the training set. A reduced set is obtained which is used instead of the training set when nearest neighbour classification is used. Prototypes are added in an incremental fashion, where at each step of the algorithm, the number of prototypes selected keeps on increasing. The number of patterns in the training data classified correctly also keeps on increasing till all patterns are classified properly. After this, a deletion operator is used where some prototypes which are not so useful are removed. This method has been used to obtain the prototypes for a variety of benchmark data sets and results have been presented. © 2001 Pattern Recognition Society. Published by Elsevier Science Ltd. All rights reserved.

Keywords: Pattern classification; Prototype selection; Supervised learning; *k*-nearest neighbour

1. Introduction

While it is a well-known fact that using nearest neighbour classification [8] gives good results and is used commonly, it suffers from various drawbacks. Firstly, this method requires a large memory space as the entire training data set has to be stored. Secondly, as it compares each test pattern with every training pattern, it requires a large computation time. Thirdly, it is sensitive to outlier samples. Prototype selection is the process by which a smaller set of patterns (prototypes) is selected and used for classification. This results in a saving in memory space, reduces the computation time and if chosen properly may result in a higher classification accuracy. Prototypes may be either elements of the training set or new patterns formed by using the training patterns.

One of the first and most popularly used methods of prototype selection is the condensed nearest-neighbour (CNN) algorithm proposed by Hart [1]. In this algorithm, first a single pattern is put in the condensed set. Then each pattern is considered and its nearest neighbour in the condensed set is found. If its label is the same as that of the pattern in the condensed set, it is left out; otherwise the new pattern is included in the condensed set. After one pass through all the training patterns, another iteration is carried out where each training pattern is classified using the patterns in the condensed set. These iterations are carried out till every training pattern is correctly classified using the NNC on the patterns in the condensed set. In 1972, Swonger proposed the iterative condensation algorithm (ICA) [2]. This approach permits both addition and deletion of samples to and from the condensed subset. It is an iterative algorithm and at every iteration there will be a reference sample vector set. Using this reference sample set, classification of the patterns in the training set is carried out. For every sample in the training set, the margin of misclassification is calculated. The sample from each class with the largest margin of misclassification is included in the reference sample vector set. At the same time, for every element in the reference vector set, if no patterns are classified correctly the element is deleted. The reduced nearest-neighbour (RNN) rule of Gates [3] derives the condensed set by iteratively contracting the given set of training samples. Each original sample is tested to see whether its

* Corresponding author. Tel.: + 91-80-309-2779; fax: + 91-80-360-2911.
E-mail address: mnm@csa.iisc.ernet.in (M.N. Murty).

deletion affects the correct classification of the remaining samples and is retained only if it has such a consequence. Sanchez et al. [4] have used proximity graphs for prototype selection. Several editing experiments are carried out by Ferri et al. [5] and comparative results presented. Soft computing tools like genetic algorithms, simulated annealing and tabu search have been used for prototype selection [6, 9, 10]. In [6], the authors have used stochastic techniques on optical character recognition data for prototype selection. This suffers from the disadvantage that we do not know the number of prototypes to use. We have given some results in Section 5 for one data set using prototype sets of different sizes.

The information available in the training patterns has to be used to classify the test patterns. If the training set has patterns close together which can be replaced by a single pattern, or if it has outliers which can be left out or if it has ambigious patterns, then reducing the set of patterns will give a higher accuracy besides reducing memory requirement and improving speed of classification. For cases where the dimensionality is high and the training set is very large, the time taken for classification can be quite significant and using a smaller set of prototypes will be highly advantageous.

The optimum number of prototypes for a particular set depends on many factors such as the number of features and the range of each feature. In a class, if there is little variation in the features, the number of prototypes required for this class may be less as compared to other classes. For example, in the digit recognition problem, the digit class "1" may not require as many prototypes as other digit classes. If the training set is already close to the optimum then prototype selection maynot give much improvement. But in data sets where the training set has a lot of redundant samples, prototype selection is highly advantageous.

2. Methodology

In any problem, especially one of high dimensionality, the boundaries of each class are very difficult to determine. In the method proposed, we make an attempt to partition the region of a class into simpler non-overlapping regions. This is done in an incremental manner, adding prototypes to a representative prototype set till finally all the training patterns are classified correctly using this set of representative prototypes. At this stage, the region pertaining to each class has been resolved into approximate Voronoi regions.

$$\mathbf{R}_j = \bigcup_{i=1}^{n} \mathbf{V}_{ji}, \quad j = 1, \ldots, c,$$

where n is the number of regions in class j. Total number of classes is c. Note that

$$\mathbf{V}_{ji} \cap \mathbf{V}_{jk} = \emptyset, \quad i \neq k$$

and

$$\mathbf{V}_{ji} \cap \mathbf{V}_{kl} = \emptyset, \quad j \neq k \text{ or } i \neq l.$$

In our algorithm, which we call the modified condensed nearest-neighbour (MCNN) algorithm, we get a set of prototypes in an incremental manner as described below. We start with a basic set of prototypes comprising one pattern from each class. The training set is classified using these prototypes. Based on the misclassified samples, a representative prototype for each class is determined and added to the set of basic prototypes. Now the training set is classified again with the augmented set of prototypes. Representative prototypes for each class are again determined based on the misclassified samples. This process is repeated till all patterns in the training set are classified correctly. Determining the representative pattern for each class for the misclassified samples is also done in an iterative manner as described later.

The method used for finding a single representative sample of a group of patterns depends on the data set used. One simple method of doing this is by using the centroid as the representative of the group of patterns. Of course, this works well in the case of patterns with a Gaussian distribution with covariance matrix equal and diagonal. This is also alright if the class can be split up into regions with the above property. In the case of binary patterns, especially, in the case of character recognition, the method described in Section 2.1 is found to work well.

2.1. Single representative sample of a group of patterns

At every iteration in the algorithm, we have to find a set of representative samples for a group of patterns. This is done by finding one sample to represent all the samples in each class.

One method of finding a single representative sample of a group of patterns is to use the sample mean or centroid of all the patterns. The sample mean or the centroid M of a collection of patterns $X_1, X_2, \ldots, X_r$ is given by

$$M = \sum_{i=1}^{r} \frac{X_i}{r},$$

where M is a vector containing f elements where f is the number of features. The sample mean gives the directional vector and the pattern in that class which is closest to M is chosen as the representative sample. This method of finding representative samples has been used in the algorithm described in Section 3.1.

While this may work well in the case of points which have a Gaussian distribution, in cases where the shape of the cluster is concentric, different classes may have identical sample means. We have used another method of finding out the typical pattern in a group of patterns. This is applicable to binary patterns only. First of all, considering a binary pattern, all the 1s are summed up to give the total T_i for the ith pattern. This is done for all the n patterns in the class and the mean no-of-ones is found out. The mean no-of-ones (MN) is

$$MN = \frac{\sum_{i=1}^{n} T_i}{n}.$$

We then find out for each feature what is the frequency of occurrence of 1. If we have n patterns $X_1, X_2, \ldots, X_n$, and if each pattern has f features, then for every feature j, we sum

$$S_j = \sum_{i=1}^{n} X_{ij}.$$

We then arrange S_j in decreasing order and form a new prototype where the first MN features occuring in the ordered sequence is set to 1 and the rest of the positions is set to zero. The vector in the training set which is closest to this prototype is found. This gives us the typical prototype for the set of patterns.

3. Modified CNN (MCNN) algorithm

3.1. Algorithm

The MCNN algorithm is detailed below.

```
1.  $\mathbf{T} = \{T_{ij}/j = 1,\ldots,c,\ i = 1,\ldots,n_j\}$,
    where $n_j$ gives the number of samples in class $j$.
2.  $t = 0$.
3.  Let $\mathbf{S}_1 = \mathbf{T}$.
4.  $t = t + 1$.
5.  $\mathbf{S}_t = \{X_{ij}/j = 1,\ldots,c,\ i = 1,\ldots,n1_j\}$
    where $n1_j =$ number of samples of class $j$ in $S_t$.
6.  $\forall j,\ j = 1,\ldots,c,\ C_{tj} = \sum_{i=1}^{n1_j} X_{ij}/n1_j$.
    /* This step finds the centroid of the misclassified samples */.
7.  $\forall j,\ j = 1,\ldots,c,\ P_{tj} = X_{kj}$
    such that $d(C_{tj}, X_{kj}) <$
    $d(C_{tj}, X_{ij})\ \forall i,\ i = 1,\ldots,n1_j,\ i \neq k$.
8.  Let $\mathbf{S}_r = \phi$.
    Let $\mathbf{S}_m = \phi$.
9.  $\forall X_{ij} \in \mathbf{S}_t$.
    if $d(X_{ij}, P_{tj}) < d(X_{ij}, P_{tk}),\ k \neq j$.
    $\mathbf{S}_r = \mathbf{S}_r \cup \{X_{ij}\}$
    else if $\exists k$ such that $d(X_{ij}, P_{tj}) > d(X_{ij}, P_{tk}),\ k \neq j$
    $\mathbf{S}_m = \mathbf{S}_m \cup \{X_{ij}\}$.
    /* If pattern is misclassified add it to $\mathbf{S}_m$. If classified correctly, add it to $\mathbf{S}_r$. */.
10. Set $\mathbf{S}_t = \mathbf{S}_r$.
11. if $\mathbf{S}_m \neq \phi$, go to 4
    /* Repeat steps 4–10 till no misclassified samples exist */.
12. $\mathbf{Q} = \mathbf{Q} \cup \{P_{ti},\ i = 1,\ldots,c\}$
    /* Add representative prototypes in $P$ to overall prototype set $Q$ */.
13. Let $\mathbf{Q} = \{Q_{ij}/i = 1,\ldots,n2_j,\ j = 1,\ldots,c\}$.
14. Let $\mathbf{S}_m = \phi$.
    Let $\mathbf{S}_r = \phi$.
15. $\forall X_{ij} \in \mathbf{T}$
    if $\exists Q_{kj}$ such that
    $d(X_{ij}, Q_{kj}) \leqslant d(X_{ij}, Q_{pq})$,
    $\forall q = 1,\ldots,c,\ q \neq j$
    and $\forall p = 1,\ldots,n2_q$,
    then $\mathbf{S}_r = \mathbf{S}_r \cup \{X_{ij}\}$,
    else $\mathbf{S}_m = \mathbf{S}_m \cup \{X_{ij}\}$.
    /* If sample is misclassified add it to $\mathbf{S}_m$. If classified correctly, add it to $\mathbf{S}_r$.*/.
16. Set $\mathbf{S}_t = \mathbf{S}_m$
17. if $\mathbf{S}_m = \phi$, stop else go to 4.
    /* if no samples are misclassified, stop and output $Q$ as the prototype set selected. */.
```

In this algorithm the symbols used are:

$\mathbf{Q}$	=	set containing the prototypes at each stage. Prototypes keep being added on to $\mathbf{Q}$ in an incremental fashion. After the termination of the algorithm $\mathbf{Q}$ contains the complete set of prototypes selected.
$\mathbf{T}$	=	Training set.
$\mathbf{S}_r$	=	set of correctly classified samples.
$\mathbf{S}_m$	=	set of misclassified samples.
c	=	number of classes.

As can be seen, steps 4–10 are repeated till we get representative samples for a group of samples which is a subset of the misclassified samples. As we get the representative samples at each stage, they are added on to the existing set of prototypes and the entire set of training patterns is classified using these prototypes. The procedure is stopped when all the patterns in the training set are classified correctly. As the above procedure is run on the training set, the number of misclassified samples keeps coming down till finally all the samples are classified correctly. We now have the set of prototypes which classifies all the training patterns correctly. It is to be noted that the number of prototypes in this set of each class may not be equal. At any iteration, if all the patterns of a particular class are classified correctly, then in the next iteration, there will not be any representative pattern added for that class. As the iterations progress, the number of classes which have representative sample added keeps on coming down. In the case of digit recognition, we found that representative samples for digit “1” were not being added quite early in the iterations, while finally, only representative samples for digit “8” were being added and all other classes were being classified correctly. Digit “1” is relatively simple and requires fewer representative samples whereas “8” being more “complex” requires more prototypes. It can be seen therefore, that this scheme identifies the "complexity" of each pattern class and accordingly choses the number of prototypes.

3.2. Analysis of MCNN algorithm

The following properties of the MCNN algorithm make it a very robust and useful algorithm.

Property 1. *The MCNN algorithm converges in a finite time.*

Proof. In steps 4–12 of algorithm MCNN, atleast one prototype is selected to be added to set **Q**. Even if one prototype is selected at each iteration, it will take at most n iterations to include all the samples in the training set to set Q. In fact, since c prototypes are added in the first pass through the algorithm, even if one sample is included from training set in the other iterations, atmost $n - c + 1$ iterations are required. Since there are finite samples in the training set, n is finite. The MCNN algorithm therefore converges in finite time. □

Property 2. *The set* **Q** *of prototypes generated by MCNN algorithm gives* 100% *accuracy on the training set.*

Proof. The prototypes P_{ij} added to the set **Q** in 12 are all samples in the training set, i.e. $P_{ij} \in \mathbf{T}$. In step 17, the termination criterion is that all $X_{ij} \in \mathbf{T}$, are correctly classified. In the worst case, if all $X_{ij} \in \mathbf{T}$ are added to the set **Q**, all samples in the training set will be classified correctly. The MCNN algorithm will therefore stop when 100% accuracy is obtained on the training set. □

Property 3. *The MCNN algorithm is order independent.*

Proof. In this algorithm, three operations are repeatedly carried out.

1. Finding the centroid or mean of a set of points (step 6).
2. Finding the closest sample to the centroid (step 7).
3. Using nearest neighbour classifier to find the samples in a data set which are classified correctly and those misclassified (steps 9 and 15).

Taking up these points one by one,

1. Consider two permutations of the data set of class j, D_{1j} and D_{2j} where the first index gives the permutation number. The sets D_{1j} and D_{2j} are the same as all and only the elements in set D_{1j} are in set D_{2j}. The order of the elements in the two sets only differ. As given in step 6, the centroid of a class j is given by

$$C_{tj} = \frac{\sum_{i=1}^{n1_j} X_{ij}}{n1_j}.$$

In this equation, the numerator is the sum of all the data elements. Addition being a commutative operation, the order of the data elements does not matter. The sum remains invariant to the order of the data elements. The denominator is the number of data elements. Thus, it can be seen that this operation is independent of the order of the data.
2. The centroid is a fixed value in the class. This operation consists of finding the distance of all the data points from the centroid. The smallest of these distances is found and that point is chosen as the representative prototype. If there is a tie between two or more points for the shortest distance from the centroid, the point with the smaller value of feature 1 is chosen. If this is the same, then the point with the smaller value of feature 2 is chosen etc. In other words, the points are chosen in lexicographic order. Here the distance measure is independent of the order of the data and the minimum of these distances remains the same whatever the order of the data. In case of a tie also, the point chosen is the same and independent of the order of the data.
3. This operation involves taking every point and finding the distance of this point from all the prototypes. The minimum of these distances is determined and the point is assigned to the appropriate class. This is done for all the points in the data set. The classification of each point is carried out and this operation is invariant of the order of the data. The points in the misclassified set S_m and the correctly classified set will also be the same. Only the order of the elements in these two sets may vary depending on the order of the data.

The other steps in the algorithm are either just assignment statements or conditional if statements or termination statements. These steps do not depend on the order of the data which is quite evident.

Thus, it can be seen that the algorithm is order independent. □

3.3. Comparison with CNN Algorithm

It is a well-known fact that CNN is order dependent. When it is applied to a particular data set, it gives a certain condensed prototype set leading to a set of non-intersecting Vornoi regions. Now if the order of the data set is changed, it will result in another set of non-intersecting Vornoi regions. If there are n data points, by permuting this data set in $n!$ ways, we can get $n!$ different condensed prototype sets. One or more of these orders of the data set lead to the condensed set obtained by the MCNN. This is evident by putting all the prototypes obtained by the MCNN in the order they are obtained at the beginning and then putting the other data points and using CNN on that. This yields the exact condensed set obtained by MCNN. Therefore,

$$\mathbf{C}_{\text{MCNN}} \subset \sum_{i=1}^{i=n!} \mathbf{C}_{i\text{CNN}},$$

where $\mathbf{C}_{\text{MCNN}}$ is the set of prototypes obtained by MCNN and $\mathbf{C}_{i\text{CNN}}$ is the set of prototypes obtained by a particular order of the data set using CNN.

Conjecture. *Does one of the n! permutations of the data produce a condensed set using CNN which is optimum?*

First of all, one measure of optimality could be the condensed set which gives 100% accuracy on the training set and has the minimum number of prototypes. It is to be noted that both CNN and MCNN give 100% accuracy on the training set. These methods are classification equivalent.

We took an example to illustrate this. Consider Fig. 1. This example consists of two classes with a total of 10 points.

The MCNN on this data yields a condensed prototype set of size 5. The points enclosed in squares in Fig. 1 are the prototypes selected by MCNN. All the 10! orders of the data set were produced and CNN was applied to them. The minimum condensed prototype set consisted of 4 prototypes. Fig. 1 shows these selected prototypes. The CNN has found a prototype set which is smaller than that of the MCNN in this case. To produce this it is necessary to run CNN for all the $n!$ permutations of the data set. Finding all permutations is very time consuming and doing it for data sets with more than about 15 data points is almost impossible. Besides, is this set of prototypes really optimum? Just by looking at Fig. 1 it can be seen that in class 1 one of the outer points is chosen. This has lead to a smaller prototype set, but it may not lead to a better classification accuracy because obviously, it does not represent the class 1 samples properly. The optimum prototype set cannot be based only on the minimum number of prototypes as in that case the set of prototypes using only the centroid in each class should be considered optimum but of course is not. What is optimum is the minimum number of prototypes which gives maximum accuracy on a validation set.

To find out the optimum condensed prototype set, another validation data set was used. This consists of 20 points, 10 belonging to class 1 and 10 belonging to class 2. This set was classified using MCNN leading to 100% accuracy. This set was then classified using all the $n!$ permutations obtained using CNN. The prototype sets which gave 100% accuracy on the validation set consisted of 5 prototypes. The MCNN condensed set was one of the sets obtained. The prototype set consisting of 4 prototypes obtained using one of the data orders of CNN gave a classification accuracy of 65%.

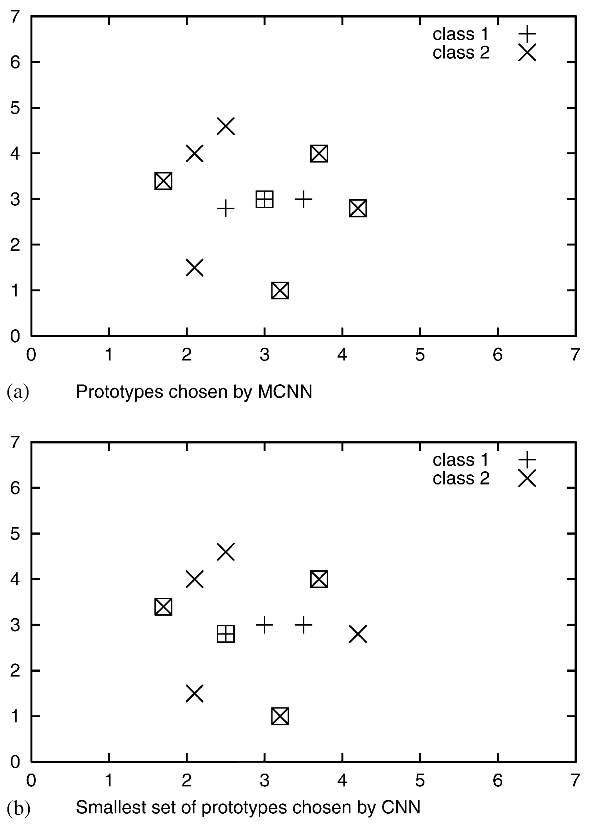

Fig. 1. A small example problem.

Let $\mathbf{O}$ be the optimal prototype set where $\mathbf{O} \subset \mathbf{T}$. The optimal set $\mathbf{O}$ should give 100% accuracy on the training set. Besides, it should be the minimum prototype set which gives higher classification accuracy on a validation set. The set $\mathbf{O}$ just like the MCNN prototypes and the CNN prototypes consists of prototypes which if put in a certain order, will be chosen by CNN as the prototype set. So if set $\mathbf{O}$ is put in the beginning of the data set in a certain order and then the rest of the data points, the optimal set $\mathbf{O}$ should be the prototypes chosen by CNN. This is one of the $n!$ different orders of the data set yielding the optimum set $\mathbf{O}$. Therefore, it can be seen that one of the $n!$ orders of the data set yield the optimum prototype set.

4. Deletion operation incorporated in the algorithm

After using the MCNN algorithm, we get a set of prototypes. The MCNN algorithm used by us, allows only addition of new prototypes to $\mathbf{Q}$. Deletion of the prototypes from $\mathbf{Q}$ is not there. Along with addition of prototypes, if deletion is also used, then the operations in the algorithm are sufficient to obtain an optimum solution. After running the MCNN and obtaining **Q**, we have assessed the usefulness of each prototype and the less useful prototypes have been deleted. This is done in the following way. The prototype set **Q** is used to classify the training set using the nearest-neighbour algorithm. This will give 100% classification. For every training pattern, the prototype from **Q** which is closest to it is noted. All prototypes which do not participate in the classification (except for classifying itself) are deleted. We thus get a reduced set of prototypes from MCNN.

Property 4. *The MCNN algorithm with the deletion operation is order independent.*

As shown in Property 3, the MCNN algorithm which produces set Q of prototypes is order independent. It is thus necessary to show that only the deletion operation is order independent. In this algorithm, each pattern in the training pattern is classified using the prototype set **Q** and each pattern closest to the training pattern from set **Q** is noted. After going through the entire training set, each prototype from set **Q** is examined and if it does not participate in the classification it is deleted. Since the deletion is carried out after going through the classification of every point in the training set, the order of data in the training set does not matter.

5. Results

Before using this method, soft computing tools were used by us [6] to obtain prototypes only for the OCR data. A total of 1500 prototypes were obtained. The results obtained were not so encouraging. This may be due to the large number of features in the OCR data. Genetic algorithms (GA), simulated annealing (SA) and tabu search (TS) were used. Due to the large size of the data the determination of the fitness function which involved finding out the classification accuracy obtained on the training set was very time consuming. Due to the large number of features, a large number of iterations will have to be used so that convergence takes place. Table 1 gives the results obtained by soft computing tools as also CNN and MCNN.

MCNN has been tried out on a number of data sets. These data set are taken from Murphy and Aha [7] except for the first one which is the optical character recognition data. This consists of 6670 training patterns and 3333 test patterns. Each pattern has 192 features. There are 10 classes, corresponding to the digits 0–9. Table 2 gives relevant details of the data sets used.

The results obtained on these data is produced in Table 3. The accuracy obtained using the MCNN, the condensed nearest neighbour (CNN) and accuracy obtained using all prototypes are given for each data set. Fig. 2 shows the number of patterns classified correctly as the iterations increase for different data sets using the MCNN algorithm. Finally, all the patterns in the training set are classified correctly.

The results obtained reveal that the MCNN does fairly well. In many cases, it gives better results than the CNN and even gives better than (or matches) the accuracy obtained using all the training set as prototypes. As can be seen from Table 3, by and large MCNN gives better results than the CNN. Only in the cases of iris data and the optical digit recognition data the CNN does slightly better than the MCNN. In the cases of thyroid data and liver data, MCNN gives higher accuracy with the reduced set than using all the prototypes. In the case of wine data, the accuracy obtained using the two methods are the same. It is found that there is a drastic reduction in the number of samples used. In the case of wine data, the reduction is from 120 to 16 which is really sharp. The accuracy obtained remains the same. In the case of iris data, while the accuracy drops slightly, there is a reduction in the number of training patterns used from 99 to 11 which is a reduction by a factor of 9. In the case of optical digit recognition, since the number of training patterns is very high (3823), the reduction of prototypes to 325 gives a very noticeable increase in the speed of classification and if the reduction of accuracy by a small amount is all

Table 1
Classification accuracy obtained using OCR data

	Accuracy (%)
GA	86.02
SA	83.77
TS	70.54
CNN	87.04
MCNN	88.0

Table 2
Details of data sets used

	No. of trg. patterns	No. of test patterns	No. of features	No. of classes
OCR	6670	3333	192	10
Wine	120	57	13	3
Thyroid	144	71	5	3
Iris	99	51	4	3
Liver	230	155	6	2
Pima	512	256	8	2
Balance	416	209	4	3
Opt. dig. rec.	3823	1797	64	10

Table 3
Accuracy obtained using the MCNN method

	MCNN		CNN		Using all prototypes	
	# of prototypes	Accuracy (%)	# of prototypes	Accuracy (%)	# of prototypes	Accuracy (%)
OCR	1527	88.0	1580	87.04	6670	92.0
Wine	14	94.74	27	94.74	120	94.74
Thyroid	18	95.77	16	91.55	144	94.37
Iris	10	92.16	16	96.08	99	94.12
Liver	136	72.17	150	64.35	230	63.48
Pima	250	67.97	266	65.23	512	69.14
Balance	157	74.64	143	71.77	416	77.03
Opt. dig. rec.	325	94.32	305	96.38	3823	98.00

right, it is definitely worth using the reduced set of prototypes.

The number of iterations necessary to obtain the prototype set in each data set using MCNN and CNN is presented in Table 4. It is true that CNN requires a significantly lesser number of iterations. But since this is carried out offline it does not matter. Of more importance is to get the optimum set of prototypes. Also if CNN has to get the optimum number of prototypes it has to find prototypes using $n!$ permutations of the data, in the worst case.

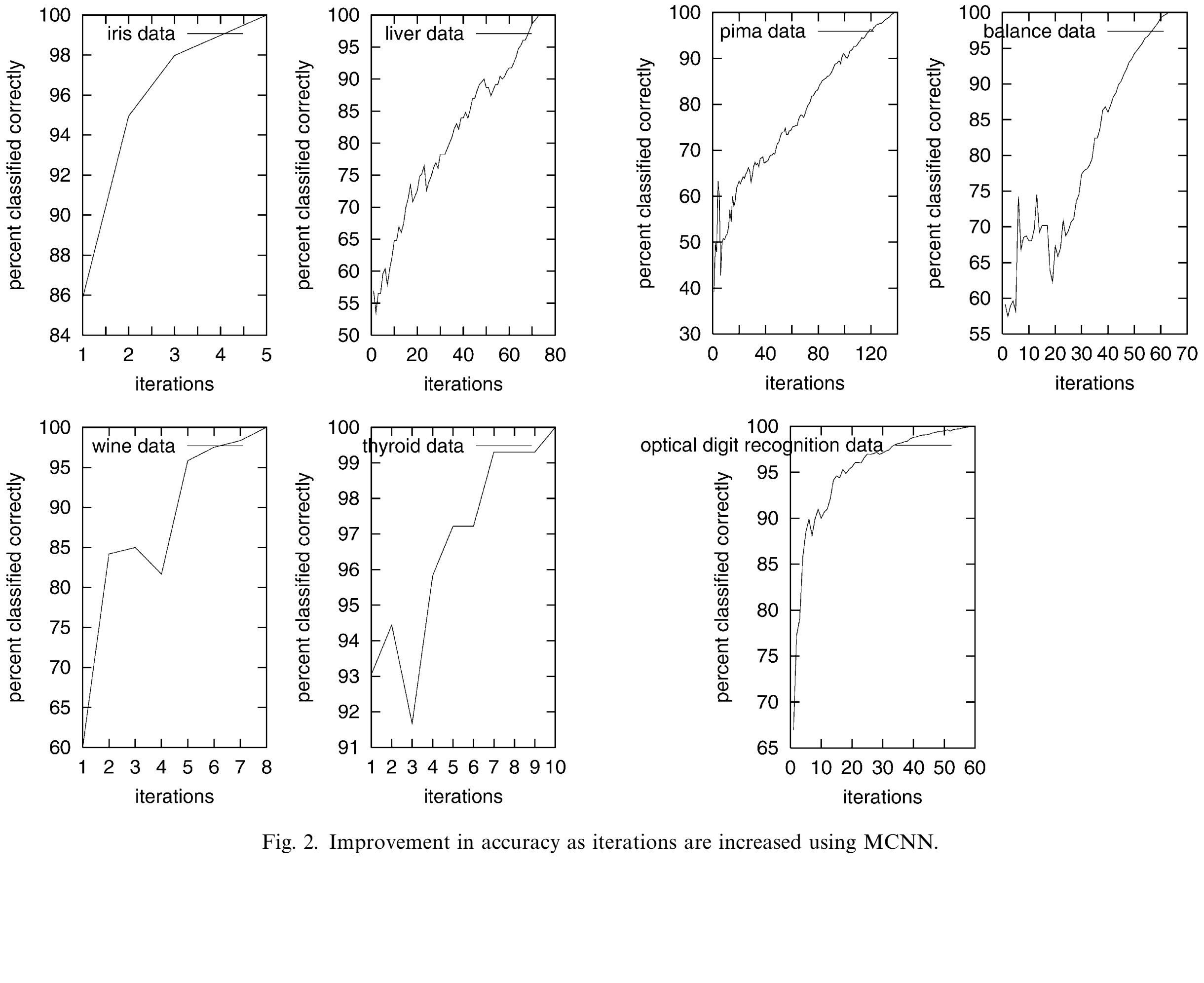

Fig. 2. Improvement in accuracy as iterations are increased using MCNN.

Table 4
No. of iterations using MCNN and CNN

	MCNN	CNN
OCR	238	5
Wine	18	5
Thyroid	12	4
Iris	6	4
Liver	73	4
Pima	138	4
Balance	63	2
Opt. dig. rec.	51	3

Regarding the template generation algorithm, the number of prototypes which will give maximum accuracy is hard to determine. Table 5 gives the results for just one data set, the wine data set, considering different sizes of the prototype set. Three techniques are used namely the GA, SA and TS. The results vary according to the number of prototypes and it is necessary to determine what are the number of prototypes required to give good classification accuracy. To determine the number of prototypes to be used of each class is a time consuming process.

After using MCNN on these data sets, the prototypes produced were reduced using the deletion operator. Table 6 gives the reduced number of prototypes and the accuracy obtained using MCNN and the CNN algorithm.

The deletion operation results in a reduction in the number of prototypes which is sometimes quite significant. In the case of Pima and Balance data, the number of prototypes reduce to half. In the liver data, the number of prototypes reduces to one third. The classification accuracy generally increases with the use of deletion operator or remains the same in a few cases.

Table 6
Accuracy obtained using the MCNN and CNN with delete operation

	MCNN		CNN	
	# of prototypes	Accuracy (%)	# of prototypes	Accuracy (%)
OCR	1098	88	1131	87.04
Wine	12	94.74	21	94.74
Thyroid	15	95.77	12	92.96
Iris	9	96.08	14	96.08
Liver	59	71.30	53	69.57
Pima	124	73.05	121	73.44
Balance	82	75.60	91	74.16
Opt. dig. rec.	290	94.49	275	96.72

6. Conclusion

This paper describes a technique for obtaining the prototypes in an incremental fashion. This technique is simple and at the end of the procedure, the training set is classified with an accuracy of 100%. It is based on taking the misclassified patterns at each stage and finding prototypes to classify these patterns correctly. The algorithm is fast and order independent. A deletion operator is also used to reduce the number of prototypes by removing prototypes which do not participate very much in the classification procedure. Results have been presented on a number of data sets. The accuracy obtained has been compared with CNN and also with using nearest-neighbour with all the training patterns.

Acknowledgements

The authors would like to thank the reviewers for doing a very thorough job of reviewing the paper and

Table 5
Accuracy obtained using template selection by GA, SA and TS for wine data

No. of prototypes	GA		SA		TS	
	No. correct Total = 57	Accuracy (%)	No. correct Total = 57	Accuracy (%)	No. correct Total = 57	Accuracy (%)
12	50	87.72	47	82.45	54	94.74
15	50	87.72	51	89.47	54	94.74
30	53	92.98	50	87.72	54	94.74
45	52	91.23	54	94.74	55	96.49
60	53	92.98	54	94.74	53	92.98
90	50	87.72	54	92.98	55	96.49

giving a constructive criticism on an earlier version of the paper which has helped to improve the paper.

References

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

[2] C.W. Swonger, Sample set condensation for a condensed nearest neighbor decision rule for pattern recognition, Front. Pattern Recognition (1972) 511–519.

[3] G.W. Gates, The reduced nearest neighbour rule, IEEE Trans. Inform. Theory IT-18 (3) (1972) 431–433.

[4] J.S. Sanchez, F. Pla, F.J. Ferri, Prototype selection for the nearest neighbour rule through proximity graphs, Pattern Recognition Lett. 18 (6) (1995) 507–513.

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

[6] V. Susheela Devi, M. Narasimha Murty, Handwritten digit recognition using soft computing tools, in: S.K. Pal, A. Ghosh, M.K. Kundu (Eds.), Soft Computing for Image Processing, Springer, Berlin, 2000.

[7] P.M. Murphy, D.W. Aha, 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, 1994.

[8] T.M. Cover, P.E. Hart, Nearest neighbor pattern classification, IEEE Trans. Inform. Theory IT-13 (1967) 21–27.

[9] L. Kuncheva, Editing for the k-nearest neighbours rule by a genetic algorithm, Pattern Recognition Lett. 16 (8) (1995) 809–814.

[10] L. Kuncheva, L.C. Jain, Nearest neighbor classifier: simultaneous editing and feature selection, Pattern Recognition Lett. 20 (1999) 1149–1156.

About the Author—V. SUSHEELA DEVI received her B.E. degree in Electrical Engineering from Bangalore University and her M.S. degree from the Department of Electrical Engineering, Indian Institute of Science, Bangalore, India. She is now working towards her Ph.D. degree at the Indian Institute of Science. Her interests include Pattern Recognition, Genetic Algorithms and Data Mining.

About the Author—M. NARASIMHA MURTY received his B.E., M.E. and Ph.D. degrees from the Indian Institute of Science, Bangalore, India. He is currently a Professor in the Department of Computer Science and Automation. His research interests include Pattern Recognition, Genetic Algorithms and Data Mining.