Prototype Based Classifier Design with Pruning

Jiang Li, Michael T. Manry, and Changhua Yu
Department of Electrical Engineering, University of Texas at Arlington
Arlington, Texas 76019.
manry@uta.edu

Abstract

An algorithm is proposed to prune the prototype vectors (prototype selection) used in a nearest neighbor classifier so that a compact classifier can be obtained with similar or even better performance. The pruning procedure is error based; a prototype will be pruned if its deletion leads to the smallest classification error increase. Also each pruning iteration is followed by one epoch of Learning Vector Quantization (LVQ) training. Simulation results show that the selected prototypes can approach optimal or near optimal locations based on the training data distribution.

Introduction

The Nearest neighbor classifier (NNC) is used for many pattern recognition applications where the underlying probability distribution of the data is unknown a priori. The behavior of the NNC is bounded by two times the optimal Bayes risk (Cover & Hart 1967). Traditional NNC stores all the known data points as labelled prototypes such that makes this algorithm prohibitive for very large database, due to the limitation of computer storage and search cost for finding the nearest neighbors of an input vector. To overcome the above challenges several techniques have been proposed by researchers. $k-d\ trees$ (Sproull & Robert 1991) and $projection$ (Papadimitriou, Christos, & Bentley 1980) can reduce the searching time for the nearest neighbors but still do not decrease storage requirements. To reduce both the memory requirements and searching time, the better way is to reduce data size under the constrict that the classification accuracy is kept similar. One can use all the training samples as initial prototypes, and improve the classifier using Editing Techniques (Penrod & Wagner 1977) followed by the Condensing algorithm (Tomek 1976) for deleting outliers and internal prototypes. The performance can be further improved by employing adaptive NNC (Geva & Sitte 1992). Instance-based learning algorithms (Aha, Dennis, & Marc 1991; Wilson & Martinez 1997) remove irrelevant instances, are similar to the condensed nearest neighbor rule. Sebban et. al. (Nock & Sebban 2001) presented a boosting-based algorithm for data reduction, by weighting and combining many of the $weak$ hypotheses into a final classifier with theoretically high accuracy for the two class problem. They extended their idea to multiclass problems (Sebban, Nock, & Lallich 2002). A similar idea was proposed by Kubat and Cooperson (Kubat & Cooperson 2000).

The above algorithms do not modify instances, but merely remove misclassified or irrelevant instances. One also can generate prototypes by modifying data instances using a clustering algorithm such as the Self-Organizing Map (SOM) (Kohonen 1995). However, some problems occur when clustering is used. Consider the artificially constructed example in Fig. 1 where the probability density functions of both inputs are uniform. It is obvious that the classification error is minimized and a perfect decision boundary is defined with only one prototype located in the middle of C_1 and two prototypes in C_2. Two challenges are: 1) the number of prototypes necessary for a classifier is difficulty to determine. The optimal number of the prototypes seems to have no direct correspondence to the probability density functions of each class; 2) the optimal placement of these prototypes is not obvious. Note that in this example the ideal locations of the prototypes are not unique, we can move them and keep the decision boundary fixed and the classification error remains unchanged. Lobo and Swiniarski (Lobo & Swiniarski 1998) pruned unnecessary prototypes using a Boolean function formalization. Even though they kept the same classification error after pruning, the remaining prototypes are no longer optimally placed. Further, the choice of the pruning candidate is not always unique. Thus the solution is not unique which may lead to different generalization capability for different pruning choices. In this paper we propose an algorithm that prunes prototypes based on the error their elimination produces. LVQ2.1 (Kohonen 1990) is used to improve the pruned classifier.

Review of LVQ2.1

Consider a training set $\{\mathbf{x}_p, i_p\}$ for NNC design, where for the pth instance, $\mathbf{x}_p \in \mathbf{R}^N$ and i_p is the integer class label associated with $\mathbf{x}_p$. N_v is the total number of instances. Assume that there are N_{tc} prototypes $\mathbf{m}_k$ that have been generated, where $1 \le k \le N_{tc}$. Each prototype is assigned a class category according the plurality vote of its members. LVQ2.1 corrects the locations of these NNC prototypes as follows: For $p = 1$ to N_v

This work was funded by the Advanced Technology Program of the state of Texas under grant 003656-0129-2001.

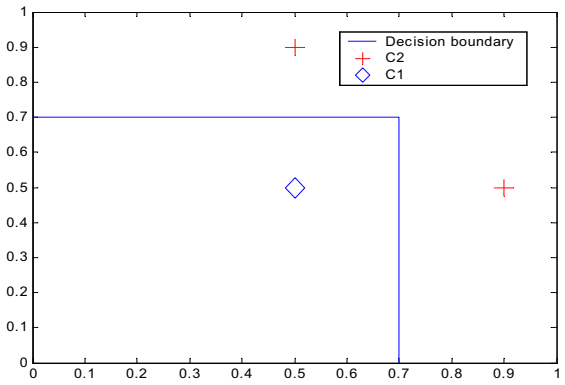

Figure 1: Uniform Data Example

1. Identify two nearest prototypes $\mathbf{m}_i$ and $\mathbf{m}_j$ to $\mathbf{x}_p$. The distance is defined as $d_k = d(\mathbf{x}_p, \mathbf{m}_k) = ||\mathbf{x}_p - \mathbf{m}_k||$, where any norm may be used. Assume the two nearest distances to $\mathbf{x}_p$ are d_i and d_j respectively.
2. If the class categories of m_i and m_j are different and one of them has the same category as $\mathbf{x}_p$, test the inequality

$$\min(\frac{d_i}{d_j}, \frac{d_j}{d_i}) > 1 - \epsilon, \tag{1}$$

   where ϵ is the "width" of the window, usually taken to be 0.35 (Kohonen 1990). Go to step 3 if this inequality is satisfied. Otherwise go to step 1.
3. If the class category of m_i is the same as that of $\mathbf{x}_p$ then update m_i and m_j as follows

$$\mathbf{m}_i \leftarrow \mathbf{m}_i + \alpha(\mathbf{x}_p - \mathbf{m}_i) \tag{2}$$
$$\mathbf{m}_j \leftarrow \mathbf{m}_j - \alpha(\mathbf{x}_p - \mathbf{m}_j) \tag{3}$$

   where $0 < \alpha < 1$ and α decrease monotonically from a starting value such as 0.1. Then go to step1.

A Weighted Distance Measure

In this section, a distance measure is introduced which can suppress random or useless features in the input vector. Training data sometimes contains inputs, which are either useless or random. When the standard Euclidean distance is used in clustering such data during NNC training, this can lead to many more prototypes than is necessary. Here we derive a weighted distance measure in the form

$$d(\mathbf{x}_p, \mathbf{m}_k) = \sum_{j=1}^{N} w(j)[x_p(j) - m_k(j)]^2 \tag{4}$$

where N is the number of inputs. For a given training set $\{\mathbf{x}_p, i_p\}$, we first design a simple classifier such as the functional link network (FLN) by minimizing

$$E = \frac{1}{N_v} \sum_{i=1}^{N_c} \sum_{p=1}^{N_v} (t_p(i) - t'_p(i))^2 \tag{5}$$

where N_c is the number of classes, $t_p(i)$ denotes the ith desired output for the pth input vector $\mathbf{x}_p$, $t'_p(i)$ denotes the ith observed output for $\mathbf{x}_p$. $t_p(i_c) = 1$ and $t_p(i_d) = 0$ where i_c denotes the correct class number for the current training input vector, and i_d denotes any incorrect class number for that vector. The ith output of the classifier for $\mathbf{x}_p$ can be written as

$$t'_p(i) = \sum_{j=1}^{N_u} w_o(i,j) X_p(j) \tag{6}$$

where $w_o(i,j)$ denotes the weight connecting the jth unit to the ith output unit. $X_p(j)$ denotes the jth basis function for the pth pattern. In an FLN, $X_p(j)$ often represents a multinomial combination of N elements of $\mathbf{x}_p$, and N_u is the number of basis functions. The following theorem provides the basis for determining useful distance measure weights from training data.

Theorem I. Let $\hat{\mathbf{t}}(\mathbf{x})$ denote the minimum mean-square error (MMSE) estimate of the desired output vector $\mathbf{t}(\mathbf{x})$. Assume that $x(j)$, the jth element of input vector $\mathbf{x}$, is statistically independent of $\mathbf{t}(\mathbf{x})$ and the other elements of the input vector. Then the derivative of $\hat{\mathbf{t}}(\mathbf{x})$ with respect to $x(j)$ is zero for all $\mathbf{x}$.

Proof: The MMSE estimate of $\mathbf{t}(\mathbf{x})$ is

$$\hat{\mathbf{t}}(\mathbf{x}) = \mathbf{E}[\mathbf{t}|\mathbf{x}] = \int_{-\infty}^{\infty} \mathbf{t} \mathbf{f}_{\mathbf{t}}(\mathbf{t}|\mathbf{x}) d\mathbf{t} \tag{7}$$

where $\mathbf{f}_{\mathbf{t}}(\mathbf{t}|\mathbf{x})$ denotes the joint probability density of the desired output vector $\mathbf{t}$ conditioned on $\mathbf{x}$. Using Bayes law,

$$\mathbf{f}_{\mathbf{t}}(\mathbf{t}|\mathbf{x}) = \frac{\mathbf{f}_{\mathbf{t},\mathbf{x}}(\mathbf{t},\mathbf{x})}{\mathbf{f}_{\mathbf{x}}(\mathbf{x})} \tag{8}$$

Letting $\mathbf{x}'$ denote $\mathbf{x}$ without the element $x(j)$,

$$\begin{aligned} \mathbf{f}_{\mathbf{t}}(\mathbf{t}|\mathbf{x}) &= \frac{\mathbf{f}_{\mathbf{t},\mathbf{x}'}(\mathbf{t},\mathbf{x}') f_{x(j)}(x(j))}{\mathbf{f}_{\mathbf{x}'}(\mathbf{x}') f_{x(j)}(x(j))} \\ &= \frac{\mathbf{f}_{\mathbf{t},\mathbf{x}'}(\mathbf{t},\mathbf{x}')}{\mathbf{f}_{\mathbf{x}'}(\mathbf{x}')} \\ &= \frac{\mathbf{f}_{\mathbf{t}}(\mathbf{t}|\mathbf{x}') \mathbf{f}_{\mathbf{x}'}(\mathbf{x}')}{\mathbf{f}_{\mathbf{x}'}(\mathbf{x}')} = \mathbf{f}_{\mathbf{t}}(\mathbf{t}|\mathbf{x}') \end{aligned}$$

Now the derivative of $\hat{\mathbf{t}}(\mathbf{x})$ with respect to $x(j)$ is

$$\begin{aligned} \frac{\partial \hat{\mathbf{t}}(\mathbf{x})}{\partial x(j)} &= \int_{-\infty}^{\infty} \frac{\partial}{\partial x(j)} [\mathbf{t} \mathbf{f}_{\mathbf{t}}(\mathbf{t}|\mathbf{x})] d\mathbf{t} \\ &= \int_{-\infty}^{\infty} \frac{\partial}{\partial x(j)} [\mathbf{t} \mathbf{f}_{\mathbf{t}}(\mathbf{t}|\mathbf{x}')] d\mathbf{t} \\ &= \mathbf{0} \end{aligned}$$

We complete the proof.

Corollary. Given the assumptions of **Theorem I**,

$$\mathbf{E}\left[\left|\left|\frac{\partial \hat{\mathbf{t}}(\mathbf{x})}{\partial x(j)}\right|\right|\right] = \mathbf{0} \tag{9}$$

where $||\cdot||$ denotes the $\mathbf{L}_1$ norm.

Now we train a FLN network, whose output for the pth pattern is denoted by $\mathbf{t}_p$. The corollary above then implies that $u(j) \simeq 0$ where

$$u(j) = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{N_c} \left| \frac{\partial t'_p(i)}{\partial x_p(j)} \right|. \tag{10}$$

As a heuristic, the distance measure's weights are determined as

$$w(j) = \frac{u(j)}{\sum_{n=1}^{N} u(n)}. \quad (11)$$

$u(j)$, which represents the importance of $x(j)$ to the network outputs, is normalized to yield $w(j)$, which is used in (4) when calculating a distance.

NNC Pruning Algorithm

In this section, we describe the NNC pruning algorithm. First a list of the algorithm steps is presented, and then explanations for those steps are given.

Algorithm Outline

Initially we start with a large number of prototypes, then prune one prototype at a time based on the additional classification error produced if it is deleted. Since these prototypes are representative of the training data, the remaining prototypes may now be at less optimal locations. We then use an LVQ2.1 epoch to refine the prototype locations. Given a training set $\{\mathbf{x}_p, i_p\}$, and the number of classes N_c, the pruning algorithm is described as follows,

1. Make the number of prototypes per class N_{pc} sufficiently large
2. Randomly initialize these $N_{tc} = N_{pc} \cdot N_c$ prototypes, and train a separate SOM network with N_{pc} prototypes for each class. Denote the number of patterns closest to the kth prototype as $N_v(k)$, where $1 \leq k \leq N_{tc}$
3. Delete the kth prototype if it does not contain any members (empty prototypes), i.e., $N_v(k) = 0$, and decrease the total number of prototypes N_{tc} by 1
4. Change the class label of a prototype if it disagrees with the plurality of the input vectors closest to it
5. Use LVQ2.1 to refine the locations of the prototypes
6. Prune one prototype based on the error increases, in case one is deleted, set $N_{tc} = N_{tc} - 1$
7. An epoch of LVQ2.1 is utilized to fine-tune the locations of the remaining prototypes
8. Go back to step 6 until the total error percentage increases 20% from the previous iteration or the total number of prototypes remaining reaches a user predefined value.

Choosing the Number of Initial Clusters

As we stated earlier, determining the exact number of prototypes required to accurately represent the data is very difficult, and there is no widely accepted method for doing this. Even though many researchers, for example Yager and Stephen (Yager & Filev 1994; Stephen 1994), have proposed such methods, the user still needs to choose some parameters to initialize the algorithms.

In our algorithm, instead of giving control parameters before training, we choose the final number of prototypes based on the testing result. It is well known that for a given finite set of training data, the training error can eventually go to zero if we keep increasing the number of prototypes. However, according to (Hughes 1968) and Vapnik (Vapnik 1995), this "overtraining" or "memorization" phenomenon decreases the generalization capability of a learning machine. Generally, on the testing result curve there is a minimum point, which indicates the *empirical minimum risk* for the given data. Let N_{pc} denote the number of prototypes per class. We initially choose a large number for N_{pc} in step 1.

Training Initial SOM Network for the Data

Given N_{pc} prototypes for each class, we then train a separate SOM network for each class in step 2. Due to the complexity of the data, there might be some prototypes that are not surrounded by data samples. These are called empty prototypes. Alternately, a prototype assigned to the ith class may have nearby data samples which belongs to the jth class. For the first case, these empty prototypes $(N_v(k) = 0)$ are deleted in step 3. For the second case the category of the prototype needs to be changed as discussed in the following section.

Changing a Prototype's Category

In order to change the categories of the prototypes in step 4, we count the members of each prototype. If a plurality of the members of a prototype belong to the ith class, for example, but the class category of that prototype initially is assigned as j, we then change the category of the prototype from j to i.

For a NNC, let T_{jk} denote the number of instances from the kth class closest to the jth prototype. Also let $i_c(j)$ denote the class category of the jth prototype. The two-dimensional array containing T_{jk} is generated by the following algorithm.

1. Set $T_{jk} = 0$ for $1 \leq j \leq N_{tc}$, $1 \leq k \leq N_c$
2. For $p = 1$ to N_v
   (a) Read $\mathbf{x}_p$ and i_p
   (b) Find j such that $d(\mathbf{x}_p, \mathbf{m}_j)$ is minimized. Let k denote the value of i_p
   (c) Accumulate patterns as $T_{jk} \leftarrow T_{jk} + 1$
3. For $j = 1$ to N_{tc}
   (a) Find k' that maximizes $T_{jk'}$
   (b) If $i_c(j) = k'$, go to 2. Otherwise change $i_c(j)$ to k'
4. Stop.

Pruning Prototypes Based on Classification Error

The goal here is to develop an algorithm to eliminate the least useful prototypes for step 6. Let k be the index of a candidate prototype to be eliminated. Then $Err(k)$ is the number of misclassified data samples after prototype k has been pruned.

1. Set $Err(k) = 0, 1 \leq k \leq N_{tc}$
2. For $p = 1$ to N_v
   (a) Identify two nearest prototypes (whose class category is l and m respectively) to input vector $\mathbf{x}_p$, and let n denote the class label of $\mathbf{x}_p$
   (b) Accumulate errors as

i. $Err(l) \leftarrow Err(l) + 1$, if $n = l, n \neq m$
ii. $Err(l) \leftarrow Err(l)$, if $n = l = m$, or $n \neq l \neq m$
iii. $Err(l) \leftarrow Err(l) - 1$, if $n \neq l, n = m$

3. Now find the smallest $Err(k)$ as $Err(k_{min})$ and eliminate the k_{min}th prototype. Note that $Err(k_{min})$ can be negative sometimes.
4. Stop.

After one prototype has been deleted, we use another epoch of LVQ2.1 to adjust the location of the remaining prototypes. This pruning process continues until the classification error increases 20% compared to the previous iteration (This number is chosen because it may indicate the remaining prototypes are much less than is necessary) or the remaining number of prototypes reaches a predefined number.

Simulation and Discussion

We study the performance of the proposed algorithm on three different data sets: uniformly distributed data, normally distributed data and data from the handwritten numeral recognition problem.

Uniformly Distributed Data

We first apply the proposed algorithm to the artificially constructed data set of Fig. 1, which contains 1000 instances. Here 400 belong to C_1 and 600 belong to C_2, and both inputs have a uniform distribution. We select the initial number of prototypes as N_{pc} = 20 for each class. The "small disks" in Fig. 2 represent prototypes initially generated by SOM, the "diamonds" represent pruned result. The 4 squares represent the situation when 4 prototypes remain. Those 4 prototypes form the decision boundary which is the dotted line in the figure. The solid line in the figure denotes the optimal decision boundary for this problem. We conclude from Fig. 2 that the proposed algorithm can find a good solution for this specific example, since the final prototypes form the optimal decision boundary and classification error is still zero. We notice that the performance will degrade a lot if we try to prune even one additional prototype. It is also observed that more prototypes for the data do not guarantee a better solution. In Fig. 2, the remaining 3 prototypes form the optimal decision boundary. However, the decision boundary (the dotted line) formed by the remaining 4 prototypes is not optimal even though the classification error is still zero.

Normally Distributed Data

The second experiment is performed on a normally distributed data set. As we illustrate in Fig. 3, the data instances from both classes follow normal distributions but with different mean and variance. We denote the instances outside the circle as class C_1 and those inside the circle as class C_2. The probability density function for the ith class is

$$f_{x,y}(x, y|C_i) = \frac{1}{2\pi\sigma_i^2} exp\{-(\frac{(x - m_{x_i})^2}{2\sigma_i^2} + \frac{(y - m_{y_i})^2}{2\sigma_i^2})\} \quad (12)$$

where $i = 1, 2$, m_{x_i}, m_{y_i} are the means for the ith class, and σ_i is the standard deviation for the ith class. In this example,

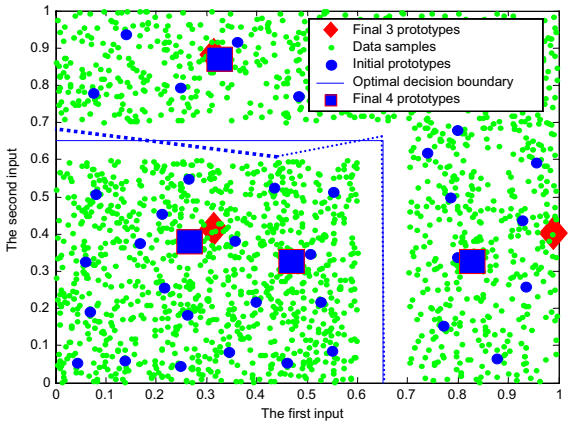

Figure 2: The Design Result for Uniformly Distributed Data

class C_1 has zero mean and $\sigma_1 = 0.5$, and the mean of class C_2 is $[m_{x_i}, m_{y_i}] = [2, 2]$ and $\sigma_2 = 1$. Each class contains 20000 instances, but only 4000 of them for each class are plotted in Fig. 3 for clarity. The optimal decision boundary calculated by Bayes decision rule for this example is

$$(x + \frac{2}{3})^2 + (y + \frac{2}{3})^2 = 4.48 \quad (13)$$

which is the circle plotted in Fig. 3. If a data instance is outside the circle, we decide it is from class C_1, otherwise it is from class C_2.

In this example we illustrate the pruning process in detail. The starting number of clusters for each class is 30. In Fig.4 we plot the pruning process when the total number of remaining prototypes N_{tc} varies from 19 to 2. The testing results corresponding to each network are listed in Table 1. The first row denotes the number of remaining prototypes, the second and the third rows represent the training and testing results (the values represent classification error in %) for the corresponding network respectively. The testing data contains the same number of instances as the training data but the two data sets have no patterns in common. Based on

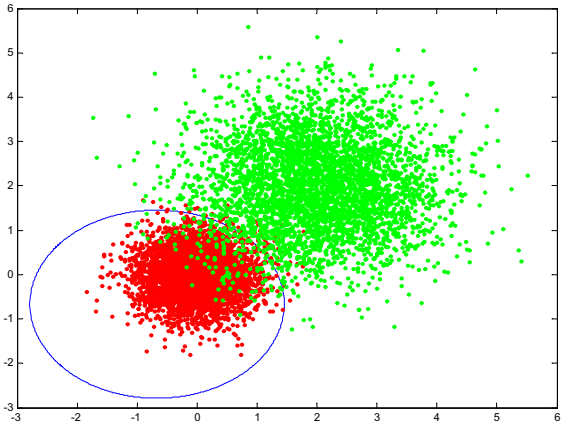

Figure 3: Normal Distributed Data

Fig. 4 and Table 1 we make the following observations.

1. The pruning process deletes more prototypes of C_1 than of C_2 in the beginning. When only one prototype remains for C_1 and 15 remain for C_2, class C_2's prototypes are deleted one by one.
2. We select the network with 5 prototypes as the final one since its testing error is lowest (see Table 1). We always prefer a simple network if we get similar generalization capability. As we can see from both Table 1 and Fig. 4, the 5 prototype network has slightly better performance than the others and it forms a near optimal decision boundary. The decision boundaries formed by the prototypes are very close to the optimal Bayes decision boundary.
3. The final prototypes we chose for the NNC classifier do not represent the probability distribution of the data. Instead, they approximate the optimal Bayes decision boundary.
4. During the pruning process, the remaining prototypes indeed approach their "optimal" locations. The number of prototypes needed for the final network is determined by the data itself. It is observed in Fig. 4 that a near optimal decision boundary is already formed when there are 14 prototypes. The pruning process just removes unneeded prototypes from class C_2. It is possible to utilize the Condensing method within the pruning process to delete internal prototypes since they do not affect class boundaries.

Under certain conditions, a Multilayer perceptron (MLP) with sigmoid activation functions in the hidden layer and trained with Back Propagation(BP) can approximate a Bayes classifier (Wan 1990). We run the BP algorithm in a three layer MLP for this normal data and compare it to the NNC classifier. Using the early stopping method (Hassoun 1995), we first determine the iteration number for a given number of hidden units and then increase the number of hidden units. The best testing performance for the MLP is observed when it has 3 hidden units with 180 training epochs. The best testing error percentage of the MLP is 2.42% for this data. Note that the theoretical Bayes error percentage for this data is 2.41%. Thus, if the data is normally distributed, both the designed NNC classifier and the MLP classifier can approach the optimal Bayes classifier. However, the data is usually not normally distributed. The NNC classifier is often employed since it is not based on any model. We thus test the designed NNC classifier on a real handwritten data set for verification.

Handwritten Numeral Data Set

The raw data consists of images from handwritten numerals collected from 3,000 people by the Internal Revenue Service. We randomly chose 300 characters from each class to generate 3,000 character training data (Gong, Yau, & Manry 1994). Images are 32 by 24 binary matrices. An image-scaling algorithm is used to remove size variation in characters. The feature set contains 16 elements. The 10 classes correspond to 10 Arabic numerals. Using the same method we generated a testing data set that contains 3000 instances.

Table 1: Training and Testing Error Percentage for Various Networks N_{tc}

Ntc	3	4	5	6	8	9	19
Traing	2.57	2.37	2.36	2.37	2.37	2.36	2.39
Testing	2.57	2.43	2.42	2.43	2.43	2.43	2.46

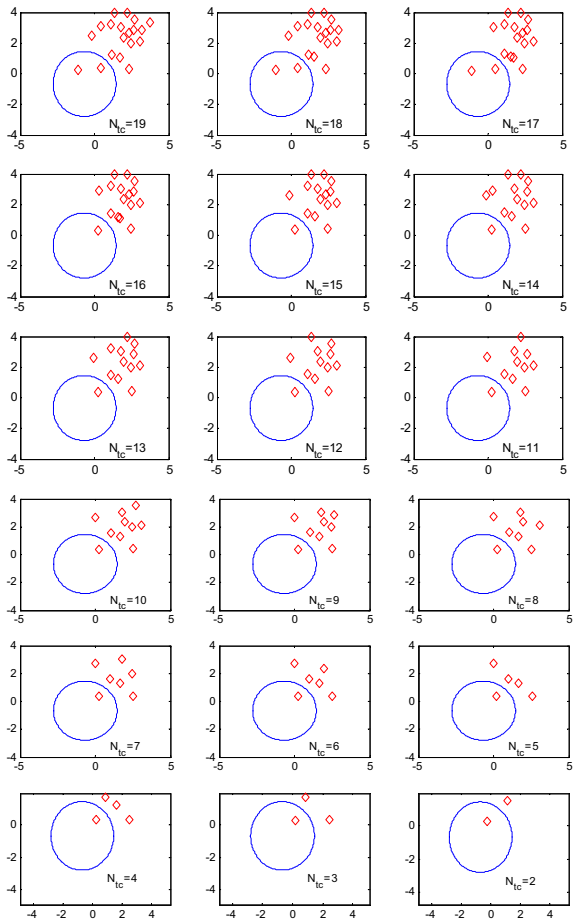

Figure 4: The Pruning process for the Normal Data

In this experiment we compare our classifier to the MLP classifier trained using the BP algorithm. For the proposed algorithm we start with 30 prototypes for each class so that the total number of prototypes is $N_{tc} = 300$. After deleting empty prototypes, 215 prototypes remain. We then prune one prototype in each iteration. The training and testing results are plotted in Fig. 5. The best testing classification error of 9.13% has been obtained when 108 prototypes remain. It is observed that if the number of prototypes increases the training error can decrease. However, the testing results become worse which indicates that over training has occurred.

To illustrate the advantages of the proposed algorithm, we compare the designed network ($N_{tc} = 108$) to a directly designed network in which only the pruning step has been eliminated. The final designed network has 109 prototypes and the testing performance is 10.47% which is worse than that of the pruned network (9.13%) even though they have almost the same number of prototypes. We investigate this difference by exploiting the prototype distribution among

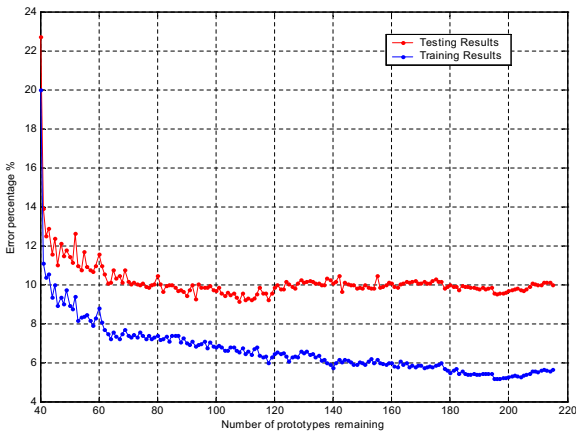

Figure 5: Training and Testing Results for Handwritten Data

Table 2: The Number of Prototypes of Each Class.

'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
7	12	13	10	13	9	9	12	11	13
4	3	9	5	8	4	18	24	18	15

classes. Table 2 shows the number of prototypes for each class, in which the first row denote the numeral image to be classified, the second and third row represent the number of prototypes in each class corresponding to the first row of the network designed without and with pruning respectively. It is observed that there are more prototypes for classes '7', '8' and '9' in the pruned network, while the prototypes are almost evenly distributed among classes for the unpruned network. Minimizing the classification error for the difficult classes '7','8' and '9' allows the pruned network to outperform the unpruned one.

For the MLP classifier we use the early stopping method to determine the number of hidden units and corresponding number of iterations needed for the network. It is found that when the number of hidden units is 20 with 550 iterations of training, we get the best testing result for this data set. We then train the network with 50 different sets of initial weights. The best testing result we get for this data is 9.87%. It is can be concluded that for this real data set, our proposed NNC classifier outperforms MLP classifier and the network designed without pruning.

Conclusion and Future Works

We have proposed a pruning algorithm for NNC prototypes. Simulation results show that the pruned NNC can be comparable to the MLP classifier for a normally distributed data set, and approaches the optimal Bayes error. We also show that it outperforms the MLP classifier for a real data set. Further work will involve applying the Condensing algorithm to the prototypes to speed up the pruning process, and comparing to existing prototype selection algorithms.

References

Aha, D. W.; Dennis, K.; and Marc, K. A. 1991. Instance-based learning algorithms. *Machine Learning* 6:37–66.

Cover, T. M., and Hart, P. E. 1967. Nearest neighbor pattern classification. *IEEE Trans. Info. Theory* 13(1):21–27.

Geva, S., and Sitte, J. 1992. Adaptive nearest neighbor pattern class. *IEEE Trans. Neur. Net.* 2(2):318–322.

Gong, W.; Yau, H. C.; and Manry, M. T. 1994. Non-gaussian feature analyses using a neural network. *Progress in Neural Networks* 2:253–269.

Hassoun, M. H. 1995. *Fundamentals of Artificial Neural Networks*. Cambridge, MA: MIT Press.

Hughes, G. F. 1968. On the mean accuracy of statistical pattern recognizers. *IEEE Transactions on Information Theory* 14:55–63.

Kohonen, T. 1990. Improved versions of learning vector quantization. *IJCNN International Joint Conference on Neural Networks* 545–550.

Kohonen, T. 1995. *Self-Organizing Maps*. Mass.: Springer-verlag.

Kubat, M., and Cooperson, M. 2000. Voting nearest-neighbor subclassifiers. *Proceedings of the Seventeenth International Conference on Machine Learning* 503–510.

Lobo, V. J., and Swiniarski, R. 1998. Pruning a classifier based on a self-organizing map using boolean function formalization. *IEEE International Joint Conference on Neural Network* 3:1910–1915.

Nock, R., and Sebban, M. 2001. Advances in adaptive prototype weighting and selection. *International Journal on Artificial Intelligence Tools* 10(1-2):137–156.

Papadimitriou; Christos, H.; and Bentley, J. L. 1980. A worst-case analysis of nearest neighbor searching by projection. *Lecture Notes in Computer Science* 85:470–482. Automata Languages and Programming.

Penrod, C., and Wagner, T. 1977. Another look at the edited nearest nearest neighbor rule. *IEEE Trans. Syst., Man, Cyber.* 7:92–94.

Sebban, M.; Nock, R.; and Lallich, S. 2002. *Journal of Machine Learning Research* 3:863–885.

Sproull, and Robert, F. 1991. Refinements to nearest neighbor searching in k-dimen. trees. *Algorithmica* 6:579–589.

Stephen, L. C. 1994. Fuzzy model identification based on vluster estimation. *Journal of Intelligent and Fuzzy Systems* 2:267–278.

Tomek, I. 1976. Two modifications of cnn. *IEEE Trans. on Syst., Man., And Cybern.* (SMC-6:):769–772.

Vapnik, V. N. 1995. *The Nature of Statistical Learning Theory*. New York: Springer-Verlag.

Wan, E. A. 1990. Neural network classification: A bayesian interpretation. *IEEE trans. on Neural Network* 1(4):303–305.

Wilson, D. R., and Martinez, T. T. 1997. Instance pruning techniques. *Proceeding of the Fourteenth International Conference on Manchine Learning* 404–411.

Yager, R., and Filev, S. P. 1994. Approximate clustering via the mountain method. *IEEE Transactions on Systems, Man and Cybernetics* 24(8):1279–1284.