Enhancing Prototype Reduction Schemes with LVQ3-Type Algorithms*

Sang-Woon Kim† and B. J. Oommen‡

Abstract

Various prototype reduction schemes have been reported in the literature. Foremost among these are the PNN, the VQ, and the SVM methods. In this paper, we shall show that these schemes can be enhanced by the introduction of a post-processing phase that is related, but not identical to, the LVQ3 process. Although the post-processing with LVQ3 has been reported for the SOM and the basic VQ methods, in this paper, we shall show that an analogous philosophy can be used in conjunction with the SVM and PNN rules. Our essential modification to LVQ3 first entails a partitioning of the respective training sets into two sets called the *Placement* set and the *Optimizing* set, which are instrumental in determining the LVQ3 parameters. Such a partitioning is novel to the literature. Our experimental results demonstrate that the proposed enhancement yields the *best* reported prototype condensation scheme to-date for both artificial data sets, and for samples involving real-life data sets.

Keywords : *Prototype Reduction, LVQ (Learning Vector Quantization), SVM (Support Vector Machines), VQ (Vector Quantization), PNN (Prototypes for Nearest Neighbor classifier), CNN (Condensed Nearest Neighbor)*

1 Introduction

1.1 Overview

In statistical pattern recognition, the nearest neighbor (NN) or the k-nearest neighbors ($k-$NN) classifiers are widely used classification rules. Each class is described using a set of sample prototypes, and the class of an unknown vector is decided based on the identity of the closest neighbor(s) which are found among all the prototypes [1]. This rule is simple, and yet it is one of the most efficient classification rules in practice.

The application of the classifier, however, often suffers from the computational complexity caused by the large number of distance computations, especially as the size increases in high dimensional problems [1], [2]. Strategies that have been proposed to solve this problem can be summarized into the three following categories: (1) Reducing

*Partially supported by the Natural Sciences and Engineering Research Council of Canada, and Myongji University, Korea.

†Member IEEE. This author can be contacted at: Div. of Computer Science and Engineering, Myongji University, Yongin, 449-728 Korea. This work was done while visiting with the School of Computer Science, Carleton University, Ottawa, Canada : K1S 5B6. e-mail address : kimsw@mju.ac.kr.

‡Senior Member IEEE.This author can be contacted at : School of Computer Science, Carleton University, Ottawa, Canada : K1S 5B6. e-mail address : oommen@scs.carleton.ca.

the size of the design set without sacrificing the performance, (2) Accelerating the computation by eliminating the necessity of calculating superfluous distances, and (3) Increasing the accuracy of the classifiers designed with the set of limited samples.

The first solution, which is the main focus of this paper, is to reduce the number of training vectors while simultaneously insisting that the classifiers built on the reduced design set perform as well, or nearly as well, as the classifiers built on the original design set. This idea has been explored for various purposes, and has resulted in the development of many algorithms. It is interesting to note that Bezdek *et al* [3], who have composed an excellent survey of the field, report that there are "zillions!" of methods for finding prototypes (see page 1459 of [3]). Our work does not compete with theirs - it merely supplements their results with a hybrid scheme which combines the salient features of a few families of the reported schemes.

Rather than re-embark on a survey of the field, we mention here a *few* representative methods of the "zillions" that have been reported. One of the first of its kind is the Condensed Nearest Neighbor (CNN) rule [4]. The CNN, however, includes "interior" samples which can be eliminated completely without changes in the performance. Accordingly, other methods have been proposed successively, such as the Reduced Nearest Neighbor (RNN) rule [5], the Prototypes for Nearest Neighbor (PNN) classifiers [6], the Selective Nearest Neighbor (SNN) rule [7], two modifications of the CNN [8], the Edited Nearest Neighbor (ENN) rule [9], and the non-parametric data reduction method [10]. Additionally, in [11], the Vector Quantization (VQ) technique was also reported as an extremely effective approach to data reduction.

The above approaches to the problem of obtaining a smaller prototype set can be categorized into two groups by considering whether or not they can *create new* prototypes, or whether they, rather, merely *select* some of the existing data points as the prototypes. The schemes reported in [6], [11] and [12] *create* new prototype vectors (and do not merely select training samples) in such a way that *these* prototypes represent all the vectors in the original set in the "best" possible manner. The methods of [4], [5], [7], [8], [9] and [10] are those in which the prototype vectors are merely *selected*. It has been proven that the former are partially superior to the latter [3, 11].

In designing NN classifiers, however, prototypes near the boundary play more important roles than those which are more interior in the feature space. In creating or selecting the prototypes, therefore, the vector points near the boundary have to be considered to be more significant, and the created prototypes need to be moved or adjusted towards the classification boundary so as to yield a higher performance. The approach that we now present is based on this philosophy, namely that of *creating, and adjusting*. Indeed, we shall first choose a reduced set of initial prototypes or code-book vectors by any of the known methods, and then learn their optimal positions with an LVQ3-type algorithm, thus minimizing the average classification error.

On the other hand, Support Vector Machines (SVM) [13] have a capability of extracting vectors that support the boundary between the two classes, and they thus satisfactorily represent the global distribution structure. Also, the learning algorithm can be easily expanded to nonlinear problems by employing techniques involving kernel functions. Thus, apart from the CNN, PNN, and VQ methods, in this paper, we also argue that the SVM can be used a means of selecting initial prototype vectors, which are subsequently operated on by LVQ3-type methods.

The paper is organized as follows: In Section 2, we briefly review representative prototype reduction methods. A complete survey is impossible here – the interested reader would find more comprehensive surveys in [2, 3]. Section 3 shows how prototype reduction methods can be enhanced using VQ-based (LVQ3) learning methods. Experiments and discussions are provided in Section 4. Finally, the conclusions are given in Section 5.

1.2 Contributions of the Paper

The main contribution of this paper is the demonstration that data condensation schemes that are of a "hybrid" sort are superior to those which are based on a *single* philosophy. Traditionally, data condensation rules utilize a variety of methods such as the CNN, the PNN, the VQ and the SVM philosophies. Also, the LVQ3 algorithm has been traditionally used in conjunction with *other VQ-type algorithms*, namely, the LVQ1 and LVQ2 modules to enhance classification [15], [16]. First of all, a minor contribution of this paper is to present a marginal enhancement of the "pure" LVQ3 algorithm. But the more important contribution is the proposed enhancement of post-processing the output of a traditional data condensation rule with this new LVQ3 rule. The result is a hybrid scheme which first creates/selects reduced prototypes, and then migrates them to enhance the potential classification. To maximize this classification, each training set is sub-divided into two subsets, which we have called the *Placement* set and the *Optimizing* set. Using these sets, the optimal parameters of the LVQ3-type algorithm are learned. The effect of this hybrid scheme is the following. While the *number* and *initial positions* of the prototypes are obtained by the data condensation rule that is used, their final positions (and consequently, the final discriminants) are learned by the mutual interactions of the *Placement* and *Optimizing* sets using LVQ3-type operations.

Another contribution of this paper is the demonstration that the SVM Classifier can be improved by invoking a 1-NN classification after the LVQ3 has operated on the support vectors. This, in our opinion, is significant, because the LVQ3 has been shown to be one of the best classifiers, especially for applications operating in a high-dimensional feature space.

The experimental results on synthetic and real-life data prove the power of these enhancements. The real-life experiments include two "medium-size" data sets, and two which involve data sets with a large number of points and a fairly high dimensionality. The results in almost all the cases is conclusive.

2 Prototype Reduction Methods

As mentioned previously, various data reduction methods have been proposed in the literature - two excellent surveys are found in [2, 3]. To put the results available in the field in the right context, we mention, in detail, the contents of the latter. The survey of [3] contains a comparison of eleven conventional PRS methods. This comparison has been performed from the view of error rates and the resultant number of prototypes that are obtained. The experiments were conducted with four experimental data sets which are both artificial and real. In summary, the eleven methods surveyed are : A combination of Wilson's ENN and Hart's CNN (W+H), the Random Selection (RS) method, Genetic Algorithms (GA), a Tabu Search (TS) scheme, a Vector Quantization-based method (LVQ1),

Decision Surface Mapping (DSM), a scheme which involves LVQ with Training Counters (LVQTC), a Bootstrap (BTS) method , a Vector Quantization (VQ) method, a Generalized LVQ-Fuzzy (GLVQ-F) scheme, and a Hard C-Means clustering (HCM) procedure. Among these, the W+H, RS, GA, and TS can be seen to be selective PRS schemes, and the others fall into the category of being creative. Additionally, the VQ, GLVQ-F, and HCM are post-supervised approaches in which the methods first find prototypes without regard to the training data labels, and then assign a class label to each prototype, while the remaining are pre-supervised ones that use the data and the class labels together to find the prototypes. Finally, the RS, LVQ1, DSM, BT, VQ, and GLVQ-F are capable of permitting the user to define the number of prototypes, while the rest of the schemes force the algorithm to decide this number.

The claim of [3], from the experimental results, is very easily stated. Based on the experimental results obtained, the authors of [3] claims that there seems to be no clear scheme that is *uniformly* superior to all the other PRS. Indeed, different methods were found to be superior for different data sets. However, the experiments showed that the *creative* methods can be superior to the selective methods, but are, typically, computationally more difficult to determine. Also, the experimental results revealed that the Pre-supervised methods are better than the Post-supervised ones. Furthermore, there seems to be no reason to believe that the auto-defined approaches are superior to the user-defined ones.

Our fundamental claim is quite simply that we can enhance *any* of the "zillions" of methods available by subjecting it to a LVQ3-type post-processing phase. From this perspective, we do not attempt to compare our new method to any single currently existing scheme. Rather, we intend to compare any enhanced scheme to its "virgin" counterpart, namely, the one that has not be subjected to the LVQ3 processing. Essentially, we submit that similar enhancements can be used to enrich any of the methods that have been currently proposed.

Rather than survey the entire field here, we attempt to review some of the most pertinent families. The CNN and the SVM are chosen as representative schemes of *selecting* methods - the former is one of first methods proposed, and the latter is historically recent. As opposed to these, the PNN and VQ (or SOM) are considered to fall within the family of prototype-*creating* algorithms.

2.1 The Condensed Nearest Neighbor rule (CNN)

The CNN [4] is suggested as a rule which reduces the size of the design set, and is largely based on statistical considerations. However, the rule does not, in general, lead to a minimal consistent set - a set which contains a minimum number of samples within it to correctly classify all the remaining samples in the given set. The procedure can be formalized as follows, where the training set is given by T, and reduced prototypes are found in T_{cnn}.

1. The first sample is copied from T to T_{cnn};
2. Do the following : Increasing i by unity from 1 to the number of samples in T per epoch:
   (a) Classify each pattern $x_i \in T$ using T_{cnn} as the prototype set;

(b) If a pattern x_i is classified incorrectly then add the pattern to T_{cnn}, and go to 3.;

3. If i is not equal to the number of samples in T, then go to 2.;

4. Else the process terminates.

2.2 Prototypes for Nearest Neighbor (PNN) Classifiers

The algorithm of finding Prototypes for Nearest Neighbor classifiers (referred to as PNN, here) [6], can be stated as follows: Given a training set T, the algorithm starts with every point in T as a prototype. Initially, set A is empty and set B is equal to T. The algorithm selects an arbitrary point in B and initially assign it to A. After this, the two closest prototypes p in A and q in B of the same class are merged, successively, into a new prototype, p^*, if the merging will not degrade the classification of the patterns in T, where p^* is the weighted average of p and q. For example, if p and q are associated with weights W_p and W_q, respectively, p^* is defined as $(W_p \cdot p + W_q \cdot q)/(W_p + W_q)$, and is assigned a weight, $W_p + W_q$. Initially, every prototype has an associated weight of unity. The procedure of PNN is sketched below.

1. Copy T to B;

2. For all $q \in B$, set the weight $W_q = 1$;

3. Select a point in B, and move it from B to A;

4. MERGE = 0;

5. While B is not empty do:

   (a) Find the closest prototypes p and q from A and B, respectively;

   (b) If p's class is not equal to q's class then insert q to A and delete it from B;

   (c) Else merge p of weight W_p, and q of weight W_q , to yield p^*, where $p^* = (W_p \cdot p + W_q \cdot q)/(W_p + W_q)$. Let the classification error rate of this new set of prototypes be ε;

   - If the ε is increased then insert q to A, and delete it from B;
   - Else delete p and q from A and B, insert p^* with weight $W_p + W_q$ to A, and MERGE++;

6. If MERGE is equal to 0 then output A as the set of trained code-book vectors, and the process terminates;

7. Copy A into B and go to 3.

Bezdek and his co-authors, proposed a modification of the PNN in [19]. First of all, instead of using the *weighted* mean of the PNN to merge prototypes, they utilized the simple arithmetic mean. Secondly, the process for searching for the candidates to be merged was modified by partitioning the distance matrix into submatrices "blocked" by common labels. This modification eliminated the consideration of candidate pairs with different labels. Based on

the results obtained from experiments conducted on the Iris data set, the authors of [19] asserted that their modified form of the PNN yielded the best consistent reduced set for designing multiple-prototype classifiers[1].

2.3 Vector Quantization and the Self Organizing Map

The foundational ideas motivating VQ and the SOM are the classical concepts that have been applied in the estimation of probability density functions. Traditionally, distributions have been represented either parametrically or non- parametrically. In the former, the user generally assumes the form of the distribution function, and the parameters of the function are learned using the available data points. In pattern recognition (classification), these estimated distributions are subsequently utilized to generate the discriminant hyper-planes or hyper-quadratics, whence the classification is achieved.

As opposed to the former, in non-parametric methods, the practitioner assumes that the data must be processed in its entirety (and not just by using a functional form to represent the data). The corresponding resulting pattern recognition (classification) algorithms are generally of the nearest neighbor (or k-nearest neighbor) philosophy, and are thus computationally expensive.

The concept of VQ [14] can be perceived as one of the earliest compromises between the above two schools of thought. Rather than represent the entire data in a compressed form using only the estimates, VQ opts to represent the data in the actual feature space. However, as opposed to the non-parametric methods which use all (or a subset) of the data in the training and testing phases of classification, VQ compresses the information by representing it using a "small" set of vectors, called the code-book vectors. These code-book vectors are migrated in the **feature** domain so that they collectively represent the distribution under consideration. We shall refer to this phase as the *Intra-Regional Polarizing* phase[17] explained below.

In both VQ and the SOM the polarizing algorithm is repeatedly presented with a point x_i from the set of points of a particular class. The neurons attempt to incorporate the topological information present in x_i. This is done as follows. First of all, the closest neuron to x_i, Y_{j*}, is determined. This neuron and a group of neurons in its neighborhood, B_{j*}, are now moved in the direction of x_i. The set B_{j*} is called the "Activation Bubble". We shall presently specify how this is determined. The actual migration of the neurons is achieved by rendering the new Y_j to be a convex combination of the current Y_j and the data point x_i for all $j \in B_{j*}$. More explicitly, the updating algorithm is as follows :

$$Y_j(t+1) = \begin{cases} (1-\alpha(t))Y_j(t) + \alpha(t)x_i & \text{if} j \in B_{j*}(t) \\ Y_j(t) & \text{otherwise} \end{cases} \tag{1}$$

where 't' is the discretized (synchronized) time index.

[1]We believe that the LVQ3-based enhancement that we propose in this paper, can also be utilized to enhance the scheme proposed in [19]. We also believe that a similar enhancement can be used for the clustering-based, genetic and random search methods proposed in [20]. This is currently being investigated. The authors are grateful to Professor Jim Bezdek for the instructive discussions we had in Spain in April 2002.

This basic algorithm has two fundamental parameters, $\alpha(t)$ and the size of the bubble $B_{j*}(t)$. $\alpha(t)$ is called the adaptation constant and satisfies $0 < \alpha(t) < 1$. Kohonen and others [15], [16] recommend steadily decrementing $\alpha(t)$ linearly from unity for the initial learning phase and then switching it to small values which decrease linearly from 0.2 for the fine-tuning phase.

The activation bubble, $B_{j*}(t)$, is the parameter which makes VQ differ from the SOM. Indeed, if the size of the bubble is always set to be zero, only the closest neuron is migrated, yielding a VQ scheme. However, in the SOM, the nearest neuron and the neurons within a bubble of activation are also migrated, and it is *this* widened migration process which permits the algorithm to be both *topology preserving* and self-organizing. The size of the bubble is initially assigned to be fairly large to allow a global ordering to develop. Consequently all the neurons tend to tie themselves into a knot for a value of $\alpha(t)$ that is close to unity; they subsequently quickly disperse. Once this coarse spatial resolution is achieved, the size of the bubble is steadily decreased. Consequently only those neurons which are most relevant to the processed input point will be effected by it. Thus the ordering which has been achieved by the coarse resolution is not disturbed, but the fine tuning on this ordering is permitted.

2.4 Support Vector Machines

The SVM [13] is a new and very promising classification technique developed at the AT&T Bell Laboratories. The main motivating criterion is to separate the classes with a surface that maximizes the margin between them. It is an approximate implementation of the structural risk minimization induction principle that aims to minimize a bound on the generalization error of a model, rather than minimizing the mean square error over the training data set, which is the philosophy that empirical risk minimization methods often use.

Training an SVM requires a set of N examples. Each example consists of an input vector x_i and its label y_i. The SVM function that has to be trained with the examples contains N free parameters, the so-called positive Lagrange multipliers $\alpha_i, i = 1, \cdots, N$. Each α_i is a measure of how much the corresponding training example influences the function. Most of the examples do not affect the function, and consequently, most of the α_i are 0. To find these parameters, we have to solve a quadratic programming (QP) problem like:

$$Minimize \qquad \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i Q_{ij} \alpha_j - \sum_{i=1}^{N} \alpha_i, \tag{2}$$

$$Subject\ to \qquad 0 \leq \alpha_i \leq C, \ \sum_{i=1}^{N} y_i \alpha_i = 0, \tag{3}$$

where Q is an $N \times N$ matrix that depends on x_i, y_i and the functional form of the SVM, and C is a constant to be chosen by the user. A larger value of C corresponds to assigning a higher penalty to the errors.

Solving the QP problem provides the support vectors of the two classes, which correspond to the examples of $\alpha_i \neq 0$. Using these, we get a hyper-plane decision function $w^T x + b = 0$, which separates the positive examples

having $+1's$ as their labels from the negative examples whose labels are all -1. The weight vector and the threshold are $w = \sum_{i=1}^{N_s} \alpha_i y_i x_i$ and $b = \frac{1}{2}(w^T x_p + w^T x_n)$, respectively, where N_s is the number of support vectors, w^T is the transpose of w, and x_p and x_n are support vectors of the positive and the negative classes, respectively.

Usually, to allow for much more general nonlinear decision functions, we have to first nonlinearly transform the input vectors into a high-dimensional feature space by a map ϕ, and then invoke a linear separation in *that* space. In this case, minimizing Eq. (2) requires the computation of dot products $\phi(x) \cdot \phi(y)$ in the higher-dimensional space. The expensive calculations, however, can be avoided by using a kernel function K obeying $K(x, y) = \phi(x) \cdot \phi(y)$, that can be evaluated efficiently. The kernel K includes functions such as polynomials, radial basis functions or sigmoidal functions. Details of the SVM can be found in [13], [18] and [21].

3 Enhancing with LVQ3-type Algorithms

3.1 The LVQ3 Algorithm

We had earlier discussed the principles motivating the LVQ and SOM families of algorithms, and discussed the intra-polarizing phase. In a multi-class problem, the code-book vectors for each region are subsequently migrated so as to ensure that they adequately represent their own regions and furthermore distinguish between the other regions. This phase, which we refer to as the *Inter-Regional Polarizing* phase [17], also implicitly learns the discriminant function to be used in a subsequent classification module. Note that these discriminant functions are of a nearest neighbor philosophy, except that the nearest neighbors are drawn from the set of code-book vectors (as opposed to the entire set of training samples). Thus, they drastically reduce the computational burden incurred in the testing of traditional non-parametric methods.

In LVQ3, two code-book vectors m_i and m_j, which are the two nearest neighbors to x, are simultaneously updated, where x and m_j belong to the same class, and x and m_i belong to different classes. Moreover, x must fall into a zone of values called the "window", which is defined around the mid-plane of m_i and m_j. Assume that d_i and d_j are the Euclidean distances of x from m_i and m_j, respectively. Then x is defined to fall in a window of relative width w if

$$min(\frac{di}{dj}, \frac{dj}{di}) > \left(\frac{1-w}{1+w}\right). \tag{4}$$

The updating rules for m_i and m_j ensure that the code-book vectors continue to approximate the respective class distributions and simultaneously enhance the quality of the classification boundary. These rules are:

$$\begin{aligned} m_i(t+1) &= m_i(t) - \alpha(t)[x(t) - m_i(t)], \\ m_j(t+1) &= m_j(t) + \alpha(t)[x(t) - m_j(t)]. \end{aligned} \tag{5}$$

Additionally, even when x, m_i and m_j belong to the same class, the code-book vectors are adjusted to enhance the improvement as follows for $k = i, j$:

$$m_k(t+1) = m_k(t) - \epsilon(t)\alpha(t)[x(t) - m_k(t)]. \tag{6}$$

In (5) and (6), t is the discretized (synchronized) time index, and $\alpha(t)$ and $\epsilon(t)$ are called the *learning rate* and *relative learning rate*, respectively.

3.2 The Proposed Data Reduction Algorithm

As mentioned earlier, the heart of the proposed algorithm involves post-processing the conventional data reduction methods using the LVQ3. This, in itself, is novel. However, the more crucial issue is that of determining the parameters of the LVQ3. We shall accomplish this by partitioning the training sets into two subsets, which are, in turn, utilized to optimize the corresponding LVQ3 parameters. We clarify all the relevant issues below.

3.2.1 Specifying the Relevant LVQ3 Criteria

The accuracy achievable in any classification task to which the LVQ3 is applied, and the time needed for learning depend on the following factors which are discussed in succession.

- An approximately-optimal number of code-book vectors assigned to each class and their initial values.
- The parameters, namely the learning rate, the relative learning rate, and the number of iteration steps.

Initialization of the code-book vectors: Since the class borders are represented piecewise-linearly by segments of mid planes between the code-book vectors of neighboring classes, it is recommended that the average distances between the adjacent code-book vectors should be the same on both sides of the borders [15]. To achieve this, the medians of the shortest distances between the initial code-book vectors of each class are first computed. If the distances turn out to be very different for the different classes, new code-book vectors may be added or old ones deleted from the deviating classes, and a tentative training cycle is run once. This procedure can be iterated a few times.

Learning rates: The learning rate $\alpha(t)$ is usually made to decrease monotonically with time. In [15], the rate α is decremented linearly during the training as below:

$$\alpha(t) = \alpha(0) \cdot \frac{Number\ of\ Iteration}{t + Number\ of\ Iteration}. \tag{7}$$

Also, the value of ϵ, the relative learning rate, is recommended to be between 0.1 and 0.5, and the related window width w is usually set to be a value between 0.2 and 0.3.

Number of learning steps: When the learning and test phases are alternated, the recognition accuracy is first improved until an optimum is reached. After that, when learning is continued, the accuracy starts to decrease slowly due to the so-called *overlearning* phenomenon. It is therefore recommended that the learning process be stopped after 50 to 200 times the total number of the code-book vectors [15]. However, the optimum number of iterations also depends on the input data.

3.2.2 Determining the Relevant LVQ3 Parameters

The algorithm that we propose consists of two steps. We first *select* or *create* initial prototypes by any one of the conventional reduction methods described earlier. After this selection/creation phase, we invoke a phase in which the optimal positions are learned with an LVQ3-type scheme. To achieve this, we assume that for every class, i, we are given two sets, the *Training* set, $T_{i,t}$, and *Validation* set, $T_{i,V}$.

We first partition the *training* set, $T_{i,t}$, into two subsets, called the *Placement* set, $T_{i,P}$, and the *Optimizing* set, $T_{i,O}$, where, $T_{i,t} = T_{i,P} \cup T_{i,O}$. The intention is that the *Placement* set is used to *position* the condensed prototypes using the LVQ3-type algorithm, and the parameters of the LVQ3-type algorithm are, in turn, optimized by testing the classification efficiency of the *current* placement on the Optimizing set, $T_{i,O}$. Thus, the training set plays a triple role: (a) First of all, it is used to obtain the initial condensed vectors; (b) Secondly, one portion of this set is used by the LVQ3-type algorithm to migrate the condensed vectors; (c) Finally, the other portion of the training set serves the purpose of "pseudo-testing", so as to obtain the best parameters for the LVQ3-type algorithm. Using these sets[2] the procedure is formalized as below for each class.

1. For every class, j, select an initial condensed prototype set $Y_{j,Test}$ by using any one of the reduction methods described earlier, and the entire training sets, $T_{i,t}$;

2. Set $Y_{Test} = \cup Y_{j,Test}$, which is the set of the training samples of all the classes.

3. Using Y_{Test} as the set of condensed prototype vectors, do the followings using the *Placement* sets, $T_{i,P}$, and the Optimizing sets, $T_{i,O}$ for all the classes:

   (a) Perform LVQ3 using the points in the *Placement* set, $T_{i,P}$. The parameters of the LVQ3 are spanned by considering increasing values of w from 0.0 to 0.5, in steps of Δw. The sets $Y_{j,Test}$ (for all j) and Y_{Test} are updated in the process. Select the best value w_0 after evaluating the accuracy of the classification rule on $T_{i,O}$, where the NN-classification is achieved by the adjusted Y_{Test};

   (b) Perform LVQ3 using the points in the *Placement* set, $T_{i,P}$. The parameters of the LVQ3 are spanned by considering increasing values of ϵ from 0.0 to 0.5, in steps of $\Delta\epsilon$. The sets $Y_{j,Test}$ (for all j) and Y_{Test} are updated in the process. Select the best value ϵ_0 after evaluating the accuracy of the classification rule on $T_{i,O}$, where the NN-classification is achieved by the adjusted Y_{Test};

[2]Specific distinct indices j and i are used just for ease of notation. The training sets are first specified in terms of the index j, but then the Placement and Optimizing Sets are used for every class i.

(c) Repeat the above steps with the current w_0 and ϵ_0, till the best values w^* and ϵ^* are obtained;

4. Determine the best prototype set Y_{Final} by invoking the LVQ3 η times with the data in $T_{i,P}$, and where the parameters are w^* and ϵ^*, where the "pseudo-testing" is achieved by using the Optimizing set, $T_{i,O}$.

The actual classification accuracy is obtained by testing the classifier using the final prototype set, Y_{Final}, and the original testing (validation) data points, $T_{i,V}$.

4 Experimental Results : Medium-Size Data Sets

4.1 Experimental Data

The proposed and the conventional prototype reduction methods were evaluated and compared by performing experiments on a number of design "medium-sized" data sets, both real and artificial, summarized in Table 1.

The dataset described as "Random", was generated randomly with a uniform distribution, but with irregular decision boundaries as indicated in Figure 1 (a). The irregularity of the boundary is clear. On the other hand, the datasets "Iris2" and "Ionosphere", which are real benchmark data sets, are cited from the UCI Machine Learning Repository [22]. Originally, the "Iris" dataset consists of three classes : Setosa, Versicolor, and Virginica. However, since the subset of Setosa samples is completely separated from the others, it is not difficult to classify it from the other two. Therefore, we have opted to employ a modified set "Iris2", which consisted only of the two classes, Versicolor and Virginica.

In the above data sets, all of the vectors were normalized within the range $[-1, 1]$ using their standard deviations, and the data set for class j was randomly split into two subsets, $T_{j,t}$ and $T_{j,V}$, of equal size. One of them was used for choosing initial code-book vectors and training the classifiers as explained above, and the other one was used in the validation (or testing) of the classifiers. Later, the role of these tests were interchanged. Figure 1 shows a training and a testing data set for the "Random" dataset, where the 200 vectors of each class were represented by '$*$' and '$\cdot$', respectively.

4.2 Selecting Initial Prototypes

In order to evaluate the reduction methods, we selected the initial prototypes from the datasets using the CNN, the PNN, the VQ and the SVM algorithms. First, we chose prototype vectors from the training datasets. Subsequently, the test datasets were classified with the 1-NN rule, where the chosen vectors were utilized as the code-book vectors. Finally, the experiment was repeated by exchanging the roles of the data sets.

As an example, Figure 2 shows the initial prototypes selected with the CNN, PNN, VQ and SVM methods. In the experiments, for the SVM program, we utilized a publicly-available software package [21] , where a polynomial of degree 3 was chosen as its kernel function.

Although no significant differences between the distributions of the selected vectors is obvious, in Figure 2, it is clear that the SVM method can choose an appropriate number of support vectors which occur near the boundary, and represent its structure appropriately. Table 2 tabulates the values of $Re(\cdot)$, the data reduction rates on the datasets, computed as $Re(\cdot) = \frac{|Total\ vectors| - |Chosen\ vectors|}{|Total\ vectors|}$, where $|\cdot|$ is the cardinality of the corresponding set.

Using the set of selected vectors as a representative set of the sample prototypes, which is considerably smaller than the original training dataset size, the 1-NN classification was used to test the testing datasets. Table 3 shows the classification error rates on the experiments, where the row termed of ORG shows the classification error rate when all vectors of the training dataset were used as the prototypes for classifying the test dataset. Also, the row $SVM^{\dagger}$, shows the classification error rates of the pure SVM classifier, not the 1-NN classifier, when it was biased and used a cubic polynomial as its kernel.

Originally, the SVM had been developed to be suitable for the solution of binary classification problems. Therefore, in this setting, when we are dealing with k-classes, we recommend its use as a *one-against-all* classifier, where the j^{th} classifier constructs a hyper-plane between the j^{th} class and the remaining $k-1$ classes.

4.3 Adjusting Prototypes with LVQ3

We also did numerous experiments to determine the classification error rates of the 1-NN classifiers after adjusting the initial prototypes with our modified LVQ3 algorithm. Four types of prototypes (initialized by the CNN, PNN, VQ and SVM, respectively) as shown in Figure 2, were post-processed (adjusted) by the LVQ3 algorithm. Beside the initial values and the number of code-book vectors, the parameters of the post-processing phase were determined as described earlier. In all the figures, the reported error rate was obtained as the average after repeating the classification 100 times each obtained by random presentations of the training vectors in the various epochs. As in the case of other learning problems, where the size of the sample patterns set plays a significant role, the same phenomenon is seen in the case of the LVQ3-adjusting learning phase. To overcome the small-sample problem, we utilized a simplified version of the scheme in which $|T_{i,O}| = 1$, namely, a variant of the leave-one-out method.

We report here the results of performing the experiment on three kinds of data sets. Additionally, for a complete comparison, the LVQ3 learning was also achieved with the code-book vectors initialized by Kohonen's method [15]. Figures plotting the classification error rates of LVQ3, CNN+LVQ3, PNN+LVQ3, VQ+LVQ3 and SVM+LVQ3 for the "Random", "Iris2" and "Ionosphere" datasets, respectively, show that the classification accuracy increases to a certain extent and then tends to decrease. Thus, in every case, it was expedient to invoke the LVQ3-type algorithm which learned the LVQ3 parameters by partitioning the training sets $T_{i,t}$ and using the sets $T_{i,P}$ and $T_{i,O}$, as opposed to using the straightforward LVQ3 algorithm.

The lowest classification error rates obtained from performing the experiments are summarized in Table 4. An overall comparison shows that *every method augmented by the LVQ3 performs better than an LVQ-based method alone*. Also, an overall comparison of Tables 3 and 4 shows *that the accuracy of every method is improved when augmented by the LVQ3 post-processing*. It is particularly worth mentioning that the pure SVM classifier can be

improved by utilizing a 1-NN classifier, after the LVQ3-learning has been invoked on the vectors extracted by the SVM algorithm. It should also be observed that this increase in classification accuracy is obtained without forfeiting excessive computational time.

5 Experimental Results : Large Data Sets

In order to further investigate the advantage gained by utilizing the proposed hybrid methods for *high dimensional applications*, we conducted experiments on "large-sized" data sets, namely, the "Sonar" and the "Arrhythmia", also obtained from the UCI Machine Learning Repository [22]. The data sets are summarized in Table 5.

The "Sonar" data set contains 208 vectors. Each sample vector, of two classes, has sixty attributes which are all continuous numerical values. The "Arrhythmia" data set contains 279 attributes, 206 of which are real-valued and the rest are nominal. In our experiments, the nominal features were replaced by the zeros. The aim of the pattern recognition exercise was to distinguish between the presence and absence of cardiac arrhythmia and to classify the feature into one of the 16 groups. In our case, in the interest of simplification, we merely attempted to classify the total instances into two category, namely, "normal" and "abnormal".

5.1 Experimental Results

We performed experiments to determine the classification accuracy rates of the 1-NN classifiers after adjusting the initial prototypes with our modified LVQ3 algorithm. First of all, the best values of the parameters ϵ^*, w^* and η^* were determined as described earlier with the training set, $T_{i,t}$. After that, the classification rates of the 1-NN classifiers were evaluated with the validation set, $T_{i,V}$. The experimental results of the "Sonar" and "Arrhythmia" datasets are shown in Table 6 and Table 7, respectively.

As in the case of the medium-size data sets, we partitioned the given set into two subsets. The first was used for choosing the initial prototype vectors, selecting the best parameter values, and training the classifiers. The second set was used in the validation of the classifiers. Later, the roles of these sets were interchanged. In Table 6 and Table 7, the first and second rows of the each scheme are the experimental results of the two subsets and the third row (bold-faced figures) is their averaged values. The best values of α^* and ϵ^* were determined as mentioned earlier, and those of w^* and η^* were selected by considering increasing values of w from 0.0 to 0.5, in steps of 0.01, and of η from 1000 to 10,000, in steps of 200, respectively.

From these tables we see that the philosophy of hybridizing almost uniformly gives us an enhanced performance. We first analyze the results for the "Arrhythmia" set. In this case, the classification accuracy increases in almost every scenario. Here, the classification problem is *very* difficult because of the large dimensionality, and so the advantage gained my hybridizing is not so prominent. Observe that although the basic CNN method yielded a 96.47 % accuracy, the method when enhanced with LVQ3 yielded an improved classification of 97.62 % accuracy. It should be observed that in any pattern classification problem, when the recognition accuracies are high, obtaining an *even higher* accuracy is a much more difficult task. Thus, improving the accuracy from 96 % to 97 % is often

much more difficult than increasing the accuracy from 50 % to 60 %. Enhancing the techniques with LVQ3 seems to be able to achieve these second-order effects. We observe, though, that the accuracy of the SVM method fell marginally - but we attribute this fall to the fact that the testing was done using a NN-based (not SVM-based) philosophy. The way by which we can rectify this for SVM-based testing is currently being investigated.

In the case of the "Sonar" data set, we see that the classification accuracy increases in almost every scenario - often yielding a *marked* improvement. Thus, while the basic CNN method yielded only 79.81 % accuracy, the method when enhanced with LVQ3 yielded 82.08 % accuracy. The most marked improvement was obtained for the PNN scheme, whose accuracy increased from 72.12 % to 83.08 %.

Finally, it is also observed that every method, augmented by the LVQ3, performs better than an LVQ-based method alone. This can be seen by comparing the *Acc0* and *Acc3* columns in the respective tables. The power of hybridizing is clear !

6 Conclusions

In designing nearest neighbor classifiers, prototypes near the boundary play more important roles than those which are more interior in feature space. Based on this idea, we have proposed an improved prototype reduction scheme, which first generates potential prototypes using a conventional method, and later yields superior prototypes by invoking a post-processor which is an LVQ3-type algorithm. In this paper, the initial prototype vectors were selected by using conventional methods such as the CNN, PNN, VQ and SVM. The proposed method has been tested on artificial and real-life benchmark datasets, and compared with the conventional ones. From the experimental results, we see, first of all, that 1-NN classifiers designed with the prototypes are faster and more accurate than either pure 1-NN, or the SVM classifier. In contrast to the prototypes produced by CNN, PNN, VQ and SVM, the prototypes post-processed by LVQ3 seem to better (globally and more exactly) represent the distribution of pattern examples, and also the properties that differentiate them. However, the classification error rates of the classifiers designed with the prototypes vary with the dataset. We recommend that this issue be tackled on a case-by-case basis, depending on the problem domain. We believe that the hybrid scheme presented here has powerful potential applications in data mining and text categorization.

References

[1] A. K. Jain, R. P. W. Duin and J. Mao, "Statistical pattern recognition: A review", *IEEE Trans. Pattern Anal. and Machine Intell.*, vol. PAMI-22, no. 1, pp. 4 - 37, Jan. 2000.

[2] D. V. Dasarachy, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, IEEE Computer Society Press, Los Alamitos, 1991.

[3] J. C. Bezdek and L. I. Kuncheva, "Nearest prototype classifier designs: An experimental study", *International Journal of Intelligent Systems*, vol. 16, no. 12, pp. 1445 - 1473, 2001.

[4] P. E. Hart, "The condensed nearest neighbor rule", *IEEE Trans. Inform. Theory*, vol. IT-14, pp. 515 - 516, May 1968.

[5] G. W. Gates, "The reduced nearest neighbor rule", *IEEE Trans. Inform. Theory*, vol. IT-18, pp. 431 - 433, May 1972.

[6] C. L. Chang, "Finding prototypes for nearest neighbor classifiers", *IEEE Trans. Computers*, vol. C-23, no. 11, pp. 1179 - 1184, Nov. 1974.

[7] G. L. Ritter, H. B. Woodruff, S. R. Lowry and T. L. Isenhour, "An algorithm for a selective nearest neighbor rule", *IEEE Trans. Inform. Theory*, vol. IT-21, pp. 665 - 669, Nov. 1975.

[8] I. Tomek, "Two modifcations of CNN", *IEEE Trans. Syst., Man and Cybern.*, vol. SMC-6, no. 6, pp. 769 - 772, Nov. 1976.

[9] P. A. Devijver and J. Kittler, "On the edited nearest neighbor rule", *Proc. 5th Int. Conf. on Pattern Recognition*, pp. 72 - 80, Dec. 1980.

[10] K. Fukunaga and J. M. Mantock, "Nonparametric data reduction", *IEEE Trans. Pattern Anal. and Machine Intell.*, vol. PAMI-6, no. 1, pp. 115 - 118, Jan. 1984.

[11] Q. Xie, C.A. Laszlo and R. K. Ward, "Vector quantization techniques for nonparametric classifier design", *IEEE Trans. Pattern Anal. and Machine Intell.*, vol. PAMI-15, no. 12, pp. 1326 - 1330, Dec. 1993.

[12] Y. Hamamoto, S. Uchimura and S. Tomita, "A bootstrap technique for nearest neighbor classifier design", *IEEE Trans. Pattern Anal. and Machine Intell.*, vol. PAMI-19, no. 1, pp. 73 - 79, Jan. 1997.

[13] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition", *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121 - 167, 1998.

[14] Y. Linde, A. Buzo and R. Gray, "An algorithm for vector quantizer design", *IEEE Trans. Commun.*, vol. COM-28, no. 1, pp. 84 - 95, Jan. 1980.

[15] http://cochlea.hut.fi/research/som_lvq_pak.shtml

[16] T. Kohonen, *Self-Oganizing Maps*, Berlin, Springer - Verlag, 1995.

[17] N. Aras, B. J. Oommen and I. K. Altinel, "The Kohonen network incorporating explicit statistics and its application to the travelling salesman problem", *Neural Networks*, vol. , pp. 1273 - 1284, Dec. 1999.

[18] V. N. Vapnik, *Statistical Learning Theory*, John Wiley & Sons, 1998.

[19] J. C. Bezdek, T. R. Reichherzer, G. S. Lim, and Y. Attikiouzel, "Multiple-prototype classifier design", *IEEE Trans. Systems, Man, and Cybernetics - Part C*, vol. SMC-28, no. 1, pp. 67 - 79, Feb. 1998.

[20] L. I. Kuncheva and J. C. Bezdek, "Nearest prototype classification: Clustering, genetic algorithms or random search?", *IEEE Trans. Systems, Man, and Cybernetics - Part C*, vol. SMC-28, no. 1, pp. 160 - 164, 1998.

[21] http://svm.first.gmd.de/.

[22] http://www.ics.uci.edu/mlearn/MLRepository.html.

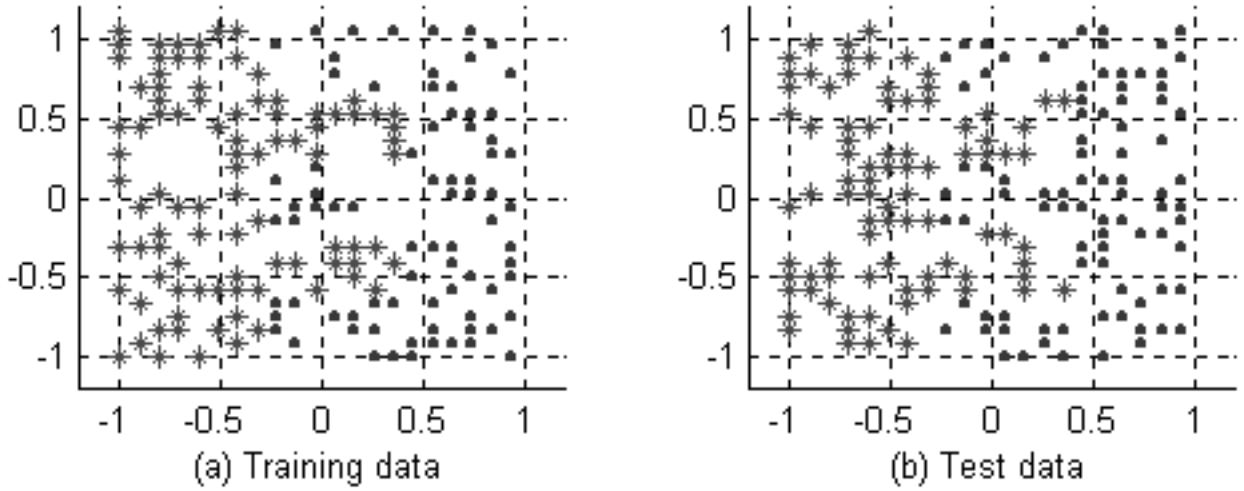

Figure 1: A training data $T_{j,t}$ (a) and a test data $T_{j,V}$ (b) for the "Random" dataset. The 200 vectors of each class are represented by '$*$' and '$\cdot$', respectively.

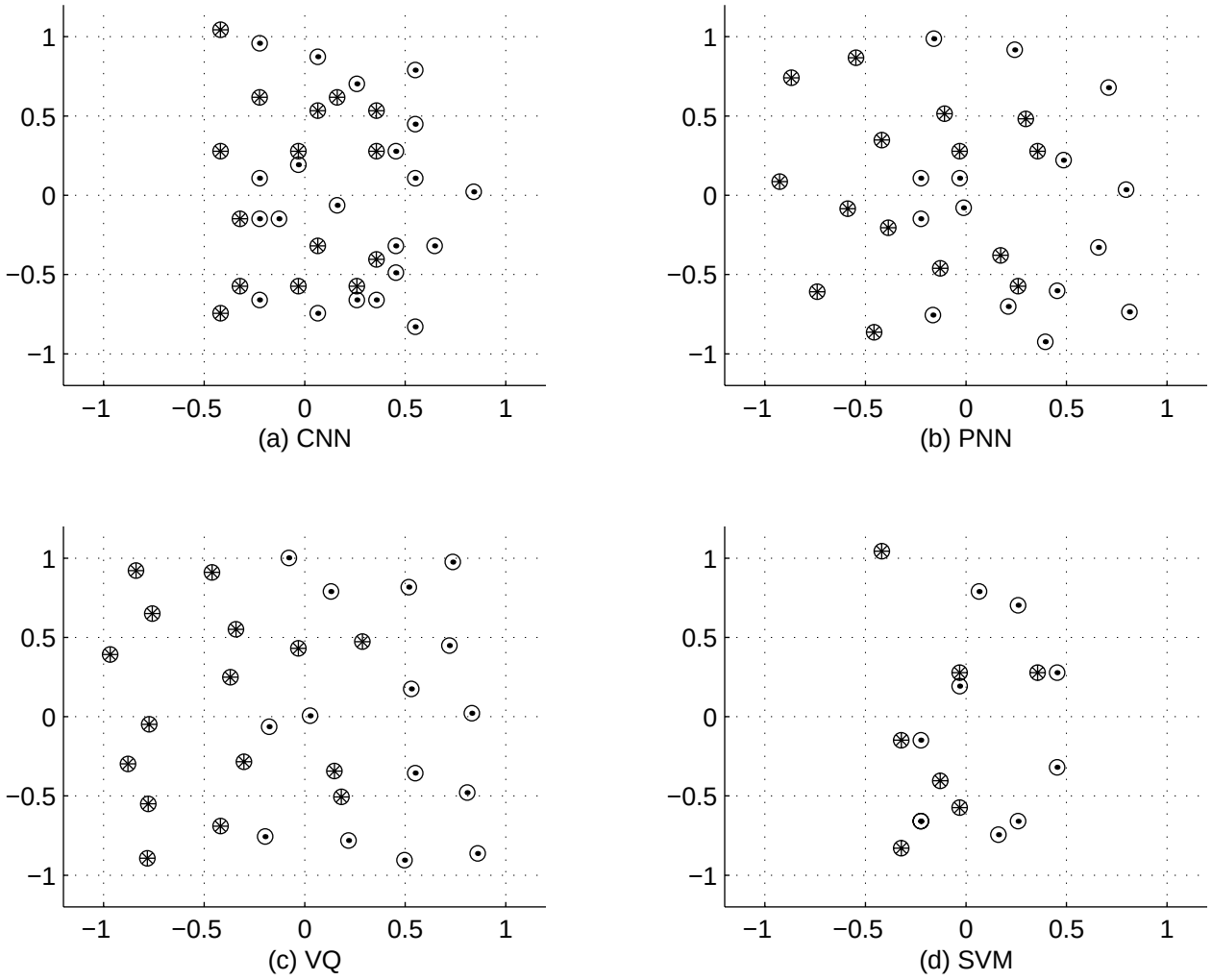

Figure 2: Initial prototypes of the "Random" dataset, which are selected by the CNN, PNN, VQ and SVM. In the pictures, selected vectors are indicated by the circled '$*$' and '$\cdot$' respectively. The numbers of selected vectors in (a), (b), (c) and (d) are 36, 30, 32 and 18, respectively.

Table 1: The benchmark data sets for experiments. The vectors are divided into two sets of equal size, and used for training and validation, alternately.

Dataset Names	Total Patterns	No. of Features	No. of Classes
Random	400 (200,200)	2	2
Iris2	100 (50,50)	4	2
Ionosphere	351 (176,175)	34	2

Table 2: The prototype compression rates of the various conventional methods on the design datasets.

Selection Methods	Random	Iris2	Ionosphere
CNN	0.83	0.71	0.74
PNN	0.86	0.83	0.80
VQ	0.84	0.68	0.98
SVM	0.92	0.85	0.80

Table 3: The classification error rates (%) on the datasets.

Selection Methods	Random	Iris2	Ionosphere
ORG	3.00	7.00	21.35
CNN	3.75	11.00	16.80
PNN	4.25	6.00	17.37
VQ	3.25	4.00	14.25
SVM†	2.75	8.00	17.37

Table 4: The classification error rates (%) of the various reported schemes after the LVQ3 post-processing.

Reduction Methods	Random	Iris2	Ionosphere
LVQ3	6.32	4.24	17.18
CNN+LVQ3	3.12	7.87	18.19
PNN+LVQ3	3.03	10.40	15.97
VQ+LVQ3	2.95	4.00	15.33
SVM+LVQ3	1.90	7.01	17.00

Table 5: The "large-sized" data sets used for experiments. The vectors are divided into two sets of equal size, and used for training and validation, alternately.

Dataset Names	Total Patterns	No. of Features	No. of Classes
Sonar	208 (104,104)	60	2
Arrhythmia	452 (226,226)	279	16

Table 6: The experimental results of the "Sonar" dataset. In these tables, *CB_size* is the size of the set of code-book vectors - which is the number of prototypes. Also, ϵ^* , w^* and η^* are the best values of the relative learning rate, the window width and the number of iteration steps, respectively. *Acc0* is the classification accuracy (%) of the initial prototype Y_{Test}, before performing the post-processing; *Acc1* and *Acc2* are the classification accuracies obtained while learning the optimal values for w^* and η^*. Finally, *Acc3*(standard deviation) is the averaged accuracy of the trained prototype Y_{Final} for the evaluation data set $T_{i,v}$.

Reduction Methods	*CB_size*	*Acc0*	ϵ^*	*Acc1*	w^*	*Acc2*	η^*	*Acc3*
LVQ3	53	69.23	0.06	83.65	0.27	83.65	1,200	77.57 (1.2014)
	59	73.08	0.06	85.58	0.10	88.46	1,200	76.60 (0.5477)
		71.16						**77.09**
CNN+ LVQ3	52	78.85	0.06	99.00	0.15	100.0	1,200	81.14 (0.9605)
	53	80.77	0.06	100.0	0.44	100.0	1,200	83.02 (0.9443)
		79.81						**82.08**
PNN+ LVQ3	34	72.12	0.06	96.15	0.19	100.0	1,400	83.56 (0.6731)
	33	72.12	0.06	93.27	0.06	98.08	1,200	82.60 (1.1971)
		72.12						**83.08**
VQ+ LVQ3	32	78.85	0.06	87.50	0.11	92.31	1,800	78.44 (1.0009)
	32	77.88	0.06	89.42	0.06	91.35	6,400	84.16 (0.9754)
		78.37						**81.30**
SVM+ LVQ3	53	82.69	0.06	95.19	0.35	95.19	5,600	80.42 (1.1091)
	59	85.58	0.06	100.0	0.32	99.04	6,400	81.29 (1.0829)
		84.14						**80.86**

Table 7: The experimental results of the "Arrhythmia" dataset. In these tables, *CB_size* is the size of the set of code-book vectors - which is the number of prototypes. Also, ϵ^* , w^* and η^* are the best values of the relative learning rate, the window width and the number of iteration steps, respectively. *Acc0* is the classification accuracy (%) of the initial prototype Y_{Test}, before performing the post-processing; *Acc1* and *Acc2* are the classification accuracies obtained while learning the optimal values for w^* and η^*. Finally, *Acc3*(standard deviation) is the averaged accuracy of the trained prototype Y_{Final} for the evaluation data set $T_{i,v}$.

Reduction Methods	*CB_size*	*Acc0*	ϵ^*	*Acc1*	w^*	*Acc2*	η^*	*Acc3*
LVQ3	65	96.90	0.06	100.0	0.36	100.0	1,000	97.12 (0.2540)
	69	97.35	0.06	98.67	0.16	98.67	1,000	98.11 (0.2175)
		97.07						**97.62**
CNN+ LVQ3	32	95.58	0.06	100.0	0.07	100.0	1,000	96.69 (0.2211)
	28	97.35	0.06	99.56	0.11	99.56	3,000	98.54 (0.3233)
		96.47						**97.62**
PNN+ LVQ3	8	98.67	0.06	100.0	0.1	100.0	5,000	98.94 (0.5663)
	7	99.56	0.06	99.56	0.17	99.56	5,000	98.86 (0.4563)
		99.12						**98.90**
VQ+ LVQ3	64	99.12	0.06	100.0	0.06	100.0	3,000	99.40 (0.2373)
	64	98.67	0.06	100.0	0.06	100.0	3,000	98.61 (0.2110)
		98.90						**99.01**
SVM+ LVQ3	65	99.56	0.06	99.56	0.17	99.56	9,000	99.07 (0.4031)
	69	99.56	0.06	99.12	0.21	99.12	6,000	98.97 (0.2414)
		99.56						**99.02**