

International Journal of Pattern Recognition and Artificial Intelligence
© World Scientific Publishing Company

Hyperspherical Prototypes for Pattern Classification

Hatem A. Fayed

*Department of Engineering Mathematics and Physics, Cairo University,
Giza, Egypt
h.fayed@eng.cu.edu.eg*

Amir F. Atiya

*Department of Computer Engineering, Cairo University,
Giza, Egypt
amir@alumni.caltech.edu*

Sherif R. Hashem

*Department of Engineering Mathematics and Physics, Cairo University,
Giza, Egypt
shashem@ids.c.gov.eg*

The nearest neighbor method is one of the most widely used pattern classification methods. However its major drawback in practice is the curse of dimensionality. In this paper we propose a new method to alleviate this problem significantly. In this method, we attempt to cover the training patterns of each class with a number of hyperspheres. The method attempts to design hyperspheres as compact as possible, and we pose this as a quadratic optimization problem. We performed several simulation experiments, and found that the proposed approach results in considerable speed-up over the k-nearest-neighbor method while maintaining the same level of accuracy. It also significantly beats other prototype classification methods (Like LVQ, RCE and CCCD) in most performance aspects.

Keywords: Pattern classification; Nearest neighbor; Hyperspherical prototypes.

1. Introduction

The nearest neighbor method is one of the oldest but still most widely used pattern classification methods^{9,11,12,14,18,19,31,35,16}. It is simple, as there is no need for uncertain or lengthy training methods. Its analytical properties in terms of performance estimates are mostly well-understood. Also, it can handle in some way non-numeric inputs, for example text inputs whereby some distance function can be defined. Consequently, it has a wide spread use in applications such as web mining. However, one of its disadvantages is the curse of dimensionality and so parametric methods outperform for large dimensions. The other major drawback is the high computational complexity for the classification stage for large data sets, as the entire training set must be stored and used for classification. It is as though

whatever one saves in terms of computational effort in the training or design stage, gets deferred to the classification stage. This speed problem is a serious drawback. We have come across many problems where there are tens of thousands of training data points. For such problems classification speed is a major issue.

The objective of this article is to propose a new method that speeds up the computation significantly. The main idea of the proposed method is to cover the data points with a number of hyperspheres. Every hypersphere will encompass a number of points from only one class, so it will be a representative of that group of points. When performing a classification, the distances to only the hypersphere centers (the “representatives”) are computed and the closest hypersphere determines the classification. Using certain developed optimization algorithms the hypersphere centers and radii are determined. As it turns out, the approach leads to significant computational savings with no sacrifice in classification performance. A preliminary version of the algorithm was introduced in Ref. 1, 2.

The paper is organized as follows. In the next section, we give a brief literature overview. The new method is described in section 3. Section 4 describes a method to split a region into two sub-regions; this is a very useful step in training. Section 5 illustrates the method that we used for classifying a query pattern. Section 6 describes the performance measures and the experimental setup. Section 7 demonstrates our experiments for synthetic and real world data sets. Finally, section 8 gives summary and concluding remarks.

2. Literature Review

There are three major approaches in the literature to speed up the nearest neighbor classification: 1) Computational methods to speed up the k-nearest neighbor search, 2) Reducing the number of data points in the nearest neighbor search (condensed nearest neighbor), 3) Clustering the space into several objects, each of them corresponds to only one class, and the class of the nearest object is assigned to the query example. Obviously, our method belongs to the third category. Here is a short overview of what is there in the literature.

2.1. *Speeding up the k-nearest neighbor search*

One way to speed up the the k-nearest neighbor search is to use a k-dimensional binary search tree ⁵. This method is very efficient when the dimension of the data space is small. Alternatively a tree can be constructed by repeating the process of dividing the points into subsets ¹⁵. Given a query point, they used a branch-and-bound search strategy to efficiently find the closest point using the tree structure. A different approach is to partition the underlying space of the sample points into a set of cells ¹⁰. A few cells located in the vicinity of the query point can be determined by calculating the distances between the query point and the centers of the cells. The nearest neighbor can then be found by searching only in these neighboring cells, instead of the whole space. A very fast algorithm was proposed to search for the

nearest neighbor within a pre-specified distance threshold³⁰. It is based on looking at one dimension at a time and excluding points that violate the given threshold. Based on triangle inequality, another elimination-based method was proposed using a number of anchor sample points to eliminate many distance calculations²⁷.

2.1.1. *Condensed nearest neighbor*

The condensed nearest neighbor¹⁷ is based on removing some points from the data set so that we speed up the search. By intelligently discarding the points that will not affect classification performance, one will end up with a smaller data set and hence faster nearest neighbor search. A number of follow-on methods have been proposed in the literature that improve on Hart's original approach. See the review in Ref. 39 for a description of the various methods. A related method (see Ref. 43 for example), is called nearest neighbor editing. It removes data points with the focus being to improve classification performance rather than search speed³⁹ (see also Ref. 3).

2.2. *Clustering the space*

A classification approach based on clustering each class region into a set of hyperspheres was first proposed in Ref. 36. This approach borrows ideas from what is called Restricted Coulomb Energy network classifiers. At each moment the system keeps some of the data points which were presented to it before, called prototypes together with a hypersphere centered around it. If a new point presented to the system is contained in some hyperspheres of inappropriate classes, the radii of the containing hyperspheres are reduced, so that none of them contains the new point. If the new point is contained in any of the hyperspheres of its own class, then no action is taken. If it is not contained in any hypersphere, then it becomes a new prototype, and is given its own hypersphere of some initial radius. Another improved version of RCE network, namely RCE-2, was proposed in Ref. 20. In this method, only the hypersphere of the closest stored pattern from a different class is modified. In Ref. 40 two modifications to the standard RCE were proposed. (1) Assigning two thresholds for the hidden units that produce two hyperspheres and determine regions of rejections. (2) Modifying the center of the hidden unit towards newly presented training examples. A variant of RCE is to cover each class region by a set of ellipsoids whose orientation coincides with the local orientation of the class region²⁵. The general learning scheme is similar to RCE's scheme³⁶. Our proposed method, even though based on the concept of hyperspheres, is very different from the RCE approaches and their variants. For example, the RCE method are sequential in nature, while in our method we consider all data points as a whole, allowing us to pose the problem as an optimization problem. The sequential nature of RCE, while giving it an adaptive nature, presents a problem for batch design in that the order of pattern presentation can have a big influence on the resulting final solution.

4 *H. Fayed , A. Atiya & S. Hashem*

As we will see in the simulation results, our methods result in significantly less number of hyperspheres (hence more compact representation) and considerably better performance than that of RCE method. Recently, a more efficient algorithm was proposed that utilizes dominating sets for class cover catch digraphs (CCCD) based on the proximity between training observations³⁴. Class cover catch digraphs are proximity graphs defined via the relationship between class-labeled observations. Each class j gives rise to a digraph D_j . The vertices of D_j are the class-conditional observations and a directed edge between two vertices exists if the proximity of the two vertices is small compared to the proximity of the vertices to the observations from the other classes. In this method, the number, location, and size of the hyperspheres are determined adaptively and its performance is usually superior to RCE methods. The difference between our methods and CCCD is that for the proposed algorithm we pose the problem of designing the hypersphere locations and centers as a quadratic optimization method. Thereby the centers are not restricted to coincide with a training pattern. On the other hand, CCCD employs an iterative method that searches among the training patterns for the best one that serves as the hypersphere center and determines its radius accordingly. In Ref. 37, a family of learning algorithms are described based on nested generalized exemplars (NGE). In NGE, an exemplar is a single data point, and a generalized exemplar is an axis-parallel hyperrectangle that may cover several data points. These hyperrectangles may overlap or nest. The NGE algorithm grows the hyperrectangles incrementally as data points are processed. Once the generalized exemplars are learned, a test data point can be classified according to the shortest Euclidean distance between the data point and the generalized exemplars. We wish to mention that the approach of clustering the space into several objects or hyperspheres has been also implemented in the unsupervised learning/clustering framework see for example Ref. 22, 41. Another approach, proposed in Ref. 13, is based on a successive splitting of the training patterns into groups, in a way such that the resulting groups will specify well-defined prototypes.

3. Smallest Covering Hyperspheres (SCHS)

The proposed approach is based on covering the data points with a number of hyperspheres. Each hypersphere covers or contains data points from only one class, and as such it is considered a prototype or a “representative”. Moreover, the hyperspheres have to be as compact as possible, in order for them to be efficient representatives. The covering procedure is performed in a sequential manner, tackling the points of one class after the other, and terminating when all points are covered. The detailed procedure is described below.

Consider a K -class pattern classification problem with N data points in a d -dimensional feature space. Consider first the data points of one of the classes (say class k). As a first step we attempt to cover all points from this class with a hypersphere that is as small as possible. This is derived by posing the problem

as a quadratic optimization problem, as follows: Denote the data points as $\mathbf{a}_i = [a_{i1} \ a_{i2} \ \dots \ a_{id}]^T$, where i indexes the data point number. The problem is to find the center $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_d]^T$ and radius r by solving :

$$\text{Min} R \quad \text{s.t.} \quad \|\mathbf{x} - \mathbf{a}_i\|_2^2 \leq R, \quad i = 1, 2, \dots, N \quad (1)$$

where $R=r^2$. The standard Lagrangian dual ⁴⁴ is:

$$\text{Min} \ \boldsymbol{\lambda}^T \mathbf{A}^T \mathbf{A} \boldsymbol{\lambda} - \mathbf{b}^T \boldsymbol{\lambda} \quad (2)$$

s.t.

$$\mathbf{e}^T \boldsymbol{\lambda} = 1 \quad (3)$$

$$\boldsymbol{\lambda} \geq \mathbf{0} \quad (4)$$

where $\mathbf{A} = [\mathbf{a}_1 \ \mathbf{a}_2 \ \dots \ \mathbf{a}_N]$, $\mathbf{b} = [\mathbf{a}_1^T \mathbf{a}_1 \ \mathbf{a}_2^T \mathbf{a}_2 \ \dots \ \mathbf{a}_N^T \mathbf{a}_N]^T$, $\mathbf{e} \in R^N$ is the vector of all ones. This is a convex optimization problem and can be easily solved by any interior-point algorithm ⁴⁵ or by the iterative barycentric coordinate descent method ²³. It was noticed that the problem of designing a smallest hypersphere to enclose a number of given points has been addressed before for some other applications such as robotics. It has been posed in Ref. 7, 21. Its extension, the smallest enclosing ellipsoid has received much attention in the literature ^{42,32,38,26}. They rely mostly on randomized algorithms or some novel nonlinear programming methods such as semidefinite programming and core-sets. In our implementation, we employed the predictor-corrector method ²⁸ in conjunction with core-sets ²⁶ to speed up the implementation especially for large data sets. By introducing Lagrange multipliers to the above problem: t as the Lagrange multiplier for the equality constraint (Eq. (3)) and \mathbf{s} as the Lagrange multiplier vector for the inequality constraints (Eq. (4)), the predictor-corrector method efficiently solves the optimality conditions:

$$2\mathbf{A}^T \mathbf{A} \boldsymbol{\lambda} - t\mathbf{e} - \mathbf{s} - \mathbf{b} = \mathbf{0} \quad (5)$$

$$\mathbf{e}^T \boldsymbol{\lambda} = 1 \quad (6)$$

$$\boldsymbol{\lambda} \geq \mathbf{0} \perp \mathbf{s} \geq \mathbf{0} \quad (7)$$

by generating iterates $(\boldsymbol{\lambda}, \mathbf{s}, t)$ that are strictly feasible with respect to the inequality constraints, that is $(\boldsymbol{\lambda}, \mathbf{s}) > \mathbf{0}$. The initial core-set is selected as follows. For the considered points, pick any point \mathbf{p}_0 , find the point \mathbf{p}_1 that is furthest from \mathbf{p}_0 , then find the point \mathbf{p}_2 that is furthest from \mathbf{p}_1 . Construct the initial core-set as $\{\mathbf{p}_1, \mathbf{p}_2\}$. Apply the predictor-corrector to find the smallest hypersphere that covers the core-set. Add the furthest point \mathbf{p}_3 from the center that lies outside the hypersphere to the core-set. Then find the smallest hypersphere that covers the new core-set $\{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3\}$. This procedure continues until all points are covered by a hypersphere.

6 *H. Fayed, A. Atiya & S. Hashem*

Algorithm 1 SCHS($\{\mathbf{a}_1^1, \mathbf{a}_2^1, \dots, \mathbf{a}_1^2, \mathbf{a}_2^2, \dots, \mathbf{a}_1^K, \mathbf{a}_2^K, \dots\}, \mu > 0, 0 < \gamma < 1$) where $\mathbf{a}_i^k \in R^d$ is the i^{th} training pattern of class k and K is the total number of classes:

```

1:  $k \leftarrow 1, Exitflag \leftarrow 0, H = \phi$  {the set of hyperspheres}.
2:  $PointSet \leftarrow \{\mathbf{a}_1^1, \mathbf{a}_2^1, \dots\}, WorkSet \leftarrow PointSet.$ 
3: while  $k < K$  do
4:   repeat
5:     Find the smallest hypersphere ( $h$ ) that encompasses  $WorkSet$ .
6:     Find points of different class that are encompassed by the hypersphere.
7:     if no such points exist then
8:        $Exitflag \leftarrow 1$ 
9:     else
10:      if number of samples of other classes encompassed by the hyper-
11:        sphere/number of samples of class  $k$  encompassed by the hypersphere
12:         $< \mu$  then
13:           $Exitflag \leftarrow 1.$ 
14:        else
15:          Compute the distance ( $d_y$ ) of the furthest point  $y$  from the center.
16:          Drop the points of  $WorkSet$  whose distances from the center are
17:          greater than or equal to  $d_y$ .
18:        end if
19:      end if
20:      until  $Exitflag = 1$ 
21:      if number of samples of class  $k$  encompassed/number of samples of the
22:         $WorkSet < \gamma$  then
23:        Split  $WorkSet$  points into two groups (according to section 4), tackle the
24:        points of each group separately (the problem is divided into two sub-
25:        problems) using the same procedure recursively by setting  $WorkSet$  to
26:        the points of the group under consideration and running from line 5.
27:      else if number of samples of class  $k$  encompassed by the hypersphere  $< N_{min}$ 
28:        then
29:        remove these samples from  $PointSet$  (considered as outliers and are dis-
30:        carded from hypersphere coverage algorithm).
31:      end if
32:       $H = H \cup h,$ 
33:      Remove points encompassed by hyperspheres assigned from  $PointSet$ .
34:      if  $PointSet$  is not empty then
35:         $WorkSet \leftarrow PointSet.$ 
36:      else if  $k < K$  then
37:         $k \leftarrow k + 1, PointSet \leftarrow \{\mathbf{a}_1^k, \mathbf{a}_2^k, \dots\}$ 
38:      end if
39:    end while

```

After the optimization steps are concluded, we shrink the radius of the resultant hypersphere to expel out all data points from different classes that lie in the hypersphere. We now have a hypersphere that contains a number of points only from class k . We have now accounted for these points, so we remove them and tackle the remaining points of class k . We repeat the same procedure for these points and keep adding hyperspheres until we have covered all points from class k . Then, we repeat the whole procedure for all other classes (see the algorithm).

We have developed a modified step that led to significant improvement in the results, and added that step to be part of the core algorithm. If at any step the resultant hypersphere encompasses too few points from *WorkSet*, then this can be overcome by splitting the patterns into two groups (see Section 4) and the process (hypersphere covering) is repeated for each group. Note that *WorkSet* is the set of points that are now targeted to be covered by the hypersphere. We also developed two more enhancements. We allow a small fraction μ of points of classes different from the considered class to be encompassed by the hypersphere generated. This will improve the generalization performance and will prevent the covering to be fragmented into too many hyperspheres. Another modification (not included in the algorithm steps displayed above) can be made to line 14 by dropping the points of *WorkSet* whose distances from the center are greater than or equal to ηd_y rather than d_y , where $0 < \eta \leq 1$ (usually η is chosen to be close to 1, for example $\eta=0.95$). This step could be repeated if some points from the other classes still exist inside the hypersphere. This modification gives the points close to the surface of the hypersphere the chance to be assigned to a perhaps better hypersphere and also speeds up the determination of the hypersphere (the innermost loop).

4. Splitting a Cluster into Two Groups

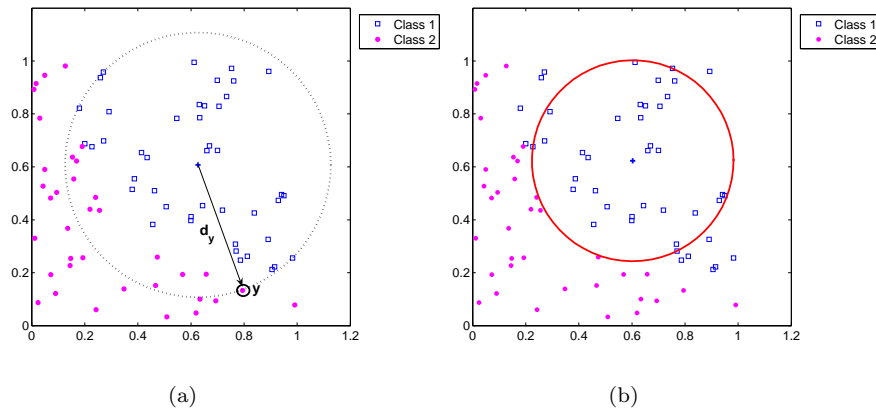


Fig. 1. Finding the first hypersphere that encompasses patterns of class 1

8 *H. Fayed, A. Atiya & S. Hashem*

During the estimation of a hypersphere in SCHS, sometimes the resultant hypersphere may encompass too few points of the *WorkSet*. This case often occurs when samples of the class under consideration are clustered into groups separated by samples of other classes, or when the hypersphere center falls in a region of class overlap. In the following time step (of obtaining the next hypersphere) we will not be much better off, because not much has changed since only a few points have been chipped off from *WorkSet*. The outcome of this is that we end up with more hyperspheres than necessary. To overcome this standoff, the samples of the *WorkSet* are divided into two groups. Consider the direction of maximum variation of the points of *WorkSet*, that is the first principal component, say α . Let $\bar{\mathbf{a}}$ be the mean vector of the points in *WorkSet*. Then break these points into the two groups: $\mathbf{z}_i^T \alpha \geq 0$ and $\mathbf{z}_i^T \alpha < 0$ where $\mathbf{z}_i = \mathbf{a}_i - \bar{\mathbf{a}}$. Then operate on each group independently.

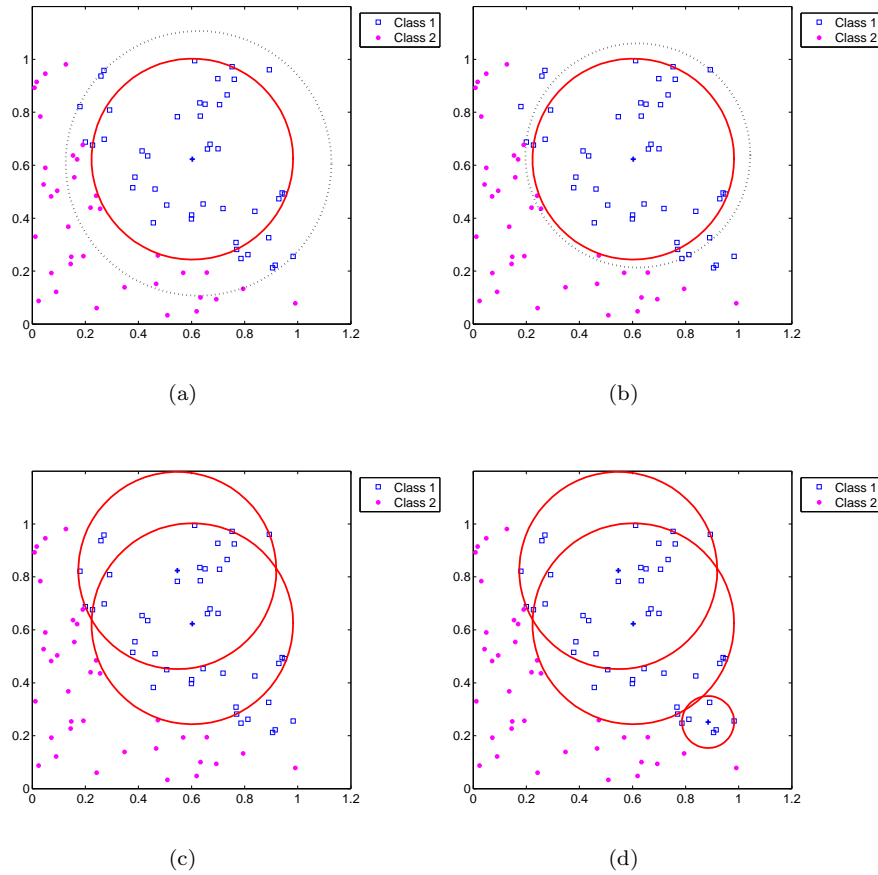


Fig. 2. Steps of finding the next hyperspheres covering patterns of class 1.

Figure 1 briefly illustrates the steps of the algorithm. Figure 1.(a) shows the hypersphere obtained through the quadratic programming formulation. Figure 1.(b) shows the final hypersphere obtained by shrinking the one obtained in Figure 1.(a) in accordance with the enhancement procedures discussed above. Figure 2 illustrates this modification. Figure 2.(a) and Figure 2.(b) show the steps of finding the next hypersphere. There are 11 points from class 1 outside the hypersphere that have to be covered in the subsequent steps. However, we found that the new hypersphere (Figure 2.(b)) hardly covers any new point. So, we split the eleven points into two groups and cover each group with a hypersphere (see Figures 2.(c) and 2.(d), the upper hypersphere and the lower small hypersphere).

5. Classification Stage

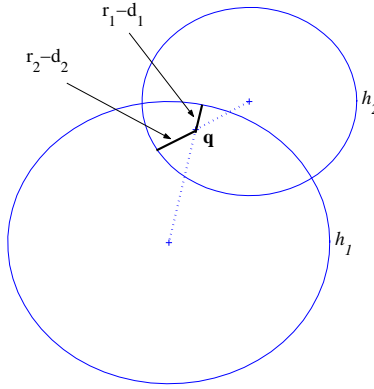


Fig. 3. Classifying a test point (\mathbf{q}) when it is encompassed by two hyperspheres. Clearly $d_2 - r_2 < d_1 - r_1$, therefore the class represented by h_2 is assigned to \mathbf{q} .

Assume that the design of the hyperspheres is complete, and that it is required to classify a given data point (\mathbf{q}). Several problems exist, for example the data point could fall inside several hyperspheres, or it could fall outside all hyperspheres. We have chosen to use the distance to the outside surface of the hypersphere (with that distance counted as negative if the point is inside the hypersphere) as the selection criterion. Specifically, we perform the following steps: Compute the distance (d_i) between the data point and the center of each hypersphere h_i . The index of the nearest neighbor hypersphere I_q is chosen as:

$$I_q = \arg \min_{i \in \{1, 2, 3, \dots, |H|\}} (d_i - r_i) \quad (8)$$

where $|H|$ is the total number of hyperspheres, r_i is the radius of hypersphere h_i . See Figure 3 for an illustration of the classification procedure in the case when a test point (\mathbf{q}) falls inside two hyperspheres.

6. Experimental Setup

To validate our methods, we used four synthetic data sets and three real world data sets. In our implementation we used MATLAB 6.5 on Windows XP operating system running on Intel PC 2.4GHZ 256MB RAM. We compared our results with the one-nearest-neighbor (1-NN), k-nearest-neighbor (K-NN), learning vector quantization (LVQ) ²⁴, restricted coulomb energy network (RCE) ³⁶, modified restricted coulomb energy network (RCE-2) ²⁰ and class cover catch digraphs (CCCD) ³⁴. Best parameters for each method are selected using 5-fold cross validation ^{8,11,14}. Suggested values for k for K-NN are $\{1,3,5,7,9,11,13,15,17,19\}$. In LVQ, the initial value of the learning rate (ϵ_0) is one of the following $\{0.01,0.05,0.1\}$ and is decreased as a proportional to the reciprocal of the iteration number; i.e. ($\epsilon_i = \epsilon_0/i$) where i is the iteration number. The number of prototypes (P_k) for class k is selected as: $\text{ceil}(\delta N_k)$ where $\delta \in \{0.01,0.1,0.2\}$, N_k is the number of training patterns whose class is k and $\text{ceil}()$ is a function that returns the nearest integer greater than or equal to its argument. Since all the training data used in our experiments are already randomized, we select the first P_k patterns for each class as the initial positions of the prototypes. To avoid overfitting, early stopping is utilized. For RCE networks, suggested values for r_{min} are: $\epsilon_1 \min_j (\max_i (a_{ij}) - \min_i (a_{ij}))$ where $i=1,2,\dots,N$ and $j=1,2,\dots,d$ and $\epsilon_1 \in \{0.001,0.01,0.1\}$ while those for r_{max} are: $\epsilon_2 \min_j (\max_i (a_{ij}) - \min_i (a_{ij}))$ where $i=1,2,\dots,N$ and $j=1,2,\dots,d$ and $\epsilon_2 \in \{0.5,0.75,1\}$. For SCHS, suggested values for γ are: 0.5,0.75, suggested values for both N_{min} and μ are: $\text{ceil}(\delta N_k)$ where $\delta \in \{0.01,0.05,0.1\}$, N_k is the number of training patterns whose class is k and suggested values for η are: 0.9,0.95. For CCCD, suggested values for α and β are similar to N_{min} , and for τ are $\{0,0.5\}$ (see Ref. 34 for the definition of each parameter). The following performance measures are used in our comparisons:

- CPU times elapsed in both training and testing. Note that the training time reported is the average over all folds and all models trained using cross validation.
- Test classification error: the percentage of misclassified patterns relative to the size of the test set.
- Number of prototypes: This is the size of training set for NN, K-NN methods, the number of generated prototypes/hyperspheres for LVQ, RCE, RCE-2, CCCD and SCHS methods.

7. Experimental Results

7.1. Synthetic data sets

7.1.1. Parabolic boundary data set (PARABOLA_05)

This data set is constructed as follows: 11,000 uniformly randomly distributed points were generated in two dimensions and two classes are assigned according

to the following predetermined parabolic boundary:

$$\begin{aligned} (x - y)^2 - \sqrt{2}(x + y) + 1 > N(0, 0.05^2) & \text{ Class 1} \\ \text{Otherwise} & \text{ Class 2} \end{aligned} \quad (9)$$

where both x and $y \in [0,1]$ and $N(0, 0.05^2)$ denotes a number generated from a normal distribution with zero mean and standard deviation of 0.05. This allows some class overlap at the boundary, which is typically expected in the majority of pattern classification problems. 10,000 points were used for testing and a varying number of training examples: 100,200,300,...,1000. Table 1 shows the training and test times for the different methods. By training we mean the process of constructing the hyperspheres for the hypersphere methods, obtaining the final prototypes for LVQ and obtaining the optimal K through cross validation for the k -nearest classifier. Table 2 shows the test classification error and the number of prototypes/hyperspheres for each method. Figure 4 shows the number of prototypes/hyperspheres of different methods while Figure 5 shows the test classification error (RCE is not plotted since it has a very bad performance compared to the others). Number of prototypes for 1-NN and K -NN are the same and equals the number of training sample; this is reported as NN in the figure.

7.1.2. *Diagonal Data Set (DIAGONAL)*

We consider a two-class problem using the distributions used in Ref. 46. Each class consists of a mixture of two normal distributions as follows:

$$\begin{aligned} p(\mathbf{x}|\omega_1) &= \frac{1}{2}N(\boldsymbol{\mu}_{11}, \mathbf{I}) + \frac{1}{2}N(\boldsymbol{\mu}_{12}, \mathbf{I}) \\ p(\mathbf{x}|\omega_2) &= \frac{1}{2}N(\boldsymbol{\mu}_{21}, \mathbf{I}) + \frac{1}{2}N(\boldsymbol{\mu}_{22}, \mathbf{I}) \end{aligned} \quad (10)$$

where $\boldsymbol{\mu}_{11} = [0 \ 0]^T, \boldsymbol{\mu}_{12} = [\mu \ \mu]^T, \boldsymbol{\mu}_{21} = [0 \ \mu]^T, \boldsymbol{\mu}_{22} = [\mu \ 0]^T$, and $N(\boldsymbol{\mu}, \mathbf{I})$ is the multivariate normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix equal to the identity matrix \mathbf{I} . We used $\mu=3.5$ in our experiments, 10,000 testing examples and varying the number of training examples: 100,200,300,...,1000. Results are shown in Table 3 and Table 4. The number of prototypes/hyperspheres and the test classification error are shown in Figure 6 and Figure 7 respectively.

7.1.3. *Interval Data Set (INTERVAL)*

It is a two-class data set generated using the distributions used in Ref. 15, 46. Each class consists of two normal distributions as for *DIAGONAL* data set with $\boldsymbol{\mu}_{11} = [0 \ 0]^T, \boldsymbol{\mu}_{12} = [6.58 \ 0]^T, \boldsymbol{\mu}_{21} = [3.29 \ 0]^T, \boldsymbol{\mu}_{22} = [9.87 \ 0]^T$. We used 10,000 testing examples and varying the number of training examples: 100,200,300,...,1000. Results are shown in Table 5 and Table 6. The number of prototypes/hyperspheres and the test classification error are shown in Figure 8 and Figure 9 respectively.

12 *H. Fayed, A. Atiya & S. Hashem*

7.1.4. *I-I Data Set (I-I)*

Here we considered a higher-dimensional data set. We generated the data points of the two classes from the n -dimensional normal distributions $N(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, $i=1,2$. The parameters are: $\boldsymbol{\mu}_1 = [0 \ 0 \ \dots \ 0]^T$, $\boldsymbol{\mu}_2 = [\mu \ 0 \ \dots \ 0]^T$, $\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 = \mathbf{I}$. The value of μ controls the degree of overlap between the two distributions. We used 6-dimensional normal distributions and set $\mu = 2.56$ in the experiments, 10,000 testing examples and varying the number of training examples: 100,200,300,...,1000. Results are shown in Table 7 and Table 8. The number of prototypes/hyperspheres and the test classification error are shown in Figure 10 and Figure 11 respectively.

7.2. *Real world data sets*

7.2.1. *Cancer data set*

In this breast cancer data set, the goal is to classify a tumor as benign or malignant. The data set consists of 9 inputs, 1 binary output and 699 examples. 65.5% of the examples are benign. 525 examples were used for training and the remaining 174 examples were used for testing. This data set was obtained from cancer1.dt^{33,6}.

7.2.2. *Diabetes data set*

This data set is about the diagnosis of diabetes of Pima Indians. The goal is to classify a Pima Indian individual as diabetes positive or negative. The data set consists of 8 inputs, 1 binary output, 768 examples. 65.1% of the examples are diabetes negative. 576 examples were used for training and the remaining 192 examples were used for testing. This data set was obtained from diabetes1.dt^{33,6}.

7.2.3. *Balance data set*

This data set was generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left or be balanced. The data set consists of 4 inputs and 3 output classes. It contains 625 examples, 288 examples are left, 288 examples are right and 49 examples are balanced. The training set consists of 468 examples, while the test set consists of 157 examples. This data set was obtained from UCI repository of machine learning databases⁶.

7.2.4. *Thyroid data set*

This data set is about the diagnosis of thyroid hypofunction. Based on patient query data and patient examination data, the task is to decide whether the patient's thyroid has overfunction, normal function, or underfunction. The data set consists of 21 inputs, 1 discrete output, 7200 examples. The class probabilities are 5.1%, 92.6% and 2.3% respectively. 5400 examples were used for training and the remaining 1800 examples were used for testing. This data set was obtained from thyroid1.dt^{33,6}.

7.2.5. *Satimage data set*

The original Landsat data for this database was generated from data purchased from NASA by the Australian Centre for Remote Sensing, and used for research at the University of New South Wales. The data set consists of 36 inputs, one discrete output of 6 classes, 6635 examples. The data were divided into a training set and a test set with 4,435 examples in the training set and 2,000 in the test set. This data set was obtained from Ref. 6 and was used in STATLOG project ²⁹.

By inspecting Table 1 through Table 10, we can notice the following:

1. The SCHS method is the slowest method in terms of training time, followed by RCE and RCE-2, then by LVQ and CCCD. K-NN is the fastest. But this is usually not a problem. It pays to spend time training or designing a classifier in return for faster classification and better performance.
2. Performance of RCE and RCE-2 degrades dramatically as the amount of noise increases in the data set. This is often due to the way they handle ambiguity of patterns enclosed by hyperspheres of different classes.
3. The smallest number of hyperspheres is generally obtained by SCHS. Naturally, this also is the best method in terms of classification speed (as classification speed is proportional to the number of hyperspheres). RCE and RCE-2 usually generate a much larger number of hyperspheres especially for noisy data sets and in overlapping regions.
4. For test-set classification performance, K-NN and SCHS were the best, followed by CCCD and LVQ with a clear margin, followed by RCE-2, followed by 1-NN, followed by RCE which was consistently worse.
5. Although CCCD's training time is considerably less than most of the methods, its testing time and performance is worse than those of SCHS. Performance of CCCD is prone to degrade with noisy boundary problems (see *PARABOLA_05's results*).

Overall, one can see that the developed method exhibit fairly good results in comparison with the existing methods. Overall, the method that provides a very good performance in most criteria (except training time) compared to the other methods is the SCHS method. It has good classification speed compared to the other hypersphere methods and is much faster than K-NN for classification, and gives good out-of-sample performance.

8. Conclusion

In this article, we presented a novel method (SCHS) for clustering class regions using hyperspheres. SCHS has some distinct features that do not exist in RCE methods. 1) Positions of hyperspheres' centers and their radii do not depend on the order of presentation of training examples to the network. 2) Storage requirements and number of training epochs are also not affected by the order of presentation

of training examples to the network. 3) Hyperspheres' centers are not restricted to be a subset of the training set. Conversely, they are learned via an optimization procedure. 4) Hyperspheres can be permitted to enclose patterns of other classes and hence it has better generalization capability (especially for noisy problems). Besides, feature 3 does not exist in CCCD method as well. Our experiments show that while not sacrificing performance, SCHS achieves a significant acceleration in the classification computation, and need smaller storage compared to the k-nearest-neighbor method. It has much better performance than the RCE method and generates significantly less number of hyperspheres.

References

1. A. Atiya, S. Hashem and H. Fayed, "New hyperspheres for pattern classification," *Proc. 1st Int. Computer Engineering Conf. (ICENCO-2004)*, Cairo, Egypt, 2004, pp. 258-263.
2. A. Atiya, S. Hashem and H. Fayed, "Pattern classification using a set of compact hyperspheres," *Proc. 1st Int. Conf. Neural Info. Proc. ICONIP'06*, Hong Kong, Springer Verlag Berlin, Heidelberg, Lecture Notes in Computer Science (4223), 2006, pp. 116-123.
3. R. Barandela, F. J. Ferri and J. S. Sanchez, "A Simple Locally Adaptive Nearest Neighbor Rule With Application To Pollution Forecasting," *Int. J. Pattern Recognition Artif. Intell.* **19** (6) (2005) 787-806.
4. M. Bazaraa and C. Shetty, *Nonlinear Programming: Theory and Algorithms*, John Wiley and Sons, New York, 1979.
5. J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM.* **18** (9) (1975) 509-517.
6. C.L. Blake, E. Keogh and C.J. Merz, "UCI repository of machine learning database", University of California, Irvine, Department of Information and Computer Science, 1994. Available: <http://www.ics.uci.edu/mlearn>.
7. L. Blumenthal and G. Wahlin, "On the spherical surface of smallest radius enclosing a bounded subset of N-dimensional Euclidean space," *Bull. Amer. Math. Soc.* **47** (1941) 771-777.
8. P. Burman, "A comparative study of ordinary cross-validation, v-fold cross-validation and repeated learning-testing methods," *Biometrika* **76** (1989) 503-514.
9. T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inform. Theory*, **13** (1967) 21-27.
10. A. Djouadi and E. Bouktache, "A fast algorithm for the nearest-neighbor classifier," *IEEE Trans. Pattern Anal. Machine Intell.* **19** (3) (1997) 277-282.
11. R. O. Duda, P. E. Hart and D.G. Stork, *Pattern Classification*, 2nd ed., John Wiley and Sons, New York, 2001.
12. S. A. Dudani, "The distance-weighted k-Nearest neighbor rule," *IEEE Trans. Sys. Man. Cybern.*, **SMC-6** (4) (1976) 325-327.
13. H. A. Fayed, S. Hashem and A. F. Atiya, "Self-generating prototypes for pattern classification," *Pattern Recognition* **40** (5) (2007) 1498-1509.
14. K. Fukunaga, *Introduction to Statistical Pattern Recognition*, 2nd Ed., Academic Press, New York, 1990.
15. K. Fukunaga and P. M. Narendra, "A branch and bound algorithm for computing k-nearest neighbors," *IEEE Trans. Comp.* **24** (1975) 750-753.

16. C. Gagn and M. Parizeau, "Coevolution of nearest neighbor classifiers," *Int. J. Pattern Recognition Artif. Intell.* **21** (5) (2007) 921–946.
17. P. Hart, "The condensed nearest neighbor rule," *IEEE Trans. Inform. Theory* **14** (1968) 515–516.
18. T. Hastie and R. Tibshirani, "Discriminant adaptive nearest neighbor classification," *IEEE Trans. Pattern Anal. Machine Intell.* **18** (6) (1996) 607–616.
19. K. Hattori and M. Takahashi, "A new nearest-neighbor rule in the pattern classification problem," *Pattern Recognition* **32** (3) (1999) 425–432.
20. M. J. Hudak, "RCE classifiers: theory and practice," *Cybern. Sys.* **23** (1992) 483–515.
21. F. John, "Extremum problems with inequalities as subsidiary conditions," *Appeared in Courant Anniversary Volume*, New York, 1948, pp. 187–204.
22. L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley and Sons, 1990.
23. L. G. Khachiyan, "Rounding of polytopes in the real number model of computation," *Math. Op. Research* **21** (1996) 307–320.
24. T. Kohonen, *Self-organization and Associative Memory*, 3rd ed., Springer-Verlag, Heidelberg, Germany, 1989.
25. M. Kositsky and S. Ullman, "Learning class regions by the union of ellipsoids," *Proc. 3th Int. Conf. Pattern Recognition (ICPR)*, IEEE Computer Society Press, 1996, pp. 750–757.
26. P. Kumar, J. S. B. Mitchell and E. A. Yildirim, "Computing core-sets and approximate smallest enclosing hyperspheres in high dimensions," *Proc. ALNEX*, 2003, pp. 45–55.
27. E. W. Lee and S.-I. Chae, "Fast design of reduced-complexity nearest-neighbor classifiers using triangular inequality," *IEEE Trans. Pattern Anal. Machine Intell.* **20** (5) (1998) 567–571.
28. S. MEHROTRA, "On the implementation of a primal-dual interior point method," *SIAM J. on Optimization.* **2** (4) (1992), 575–601.
29. D. Michie, D. J. Spiegelhalter and C. C. Taylor, *Machine Learning, Neural and Statistical Classification*, New-York, NY: Ellis Horwood, 1994.
30. S. A. Nene and S. K. Nayar, "A simple algorithm for nearest neighbor search in high dimensions," *IEEE Trans. Pattern Anal. Machine Intell.* **19** (9) (1997) 989–1003.
31. R. Nock, M. Sebban and D. Bernard, "A Simple Locally Adaptive Nearest Neighbor Rule With Application To Pollution Forecasting," *Int. J. Pattern Recognition Artif. Intell.* **17** (8) (2003) 1369–1382.
32. M. J. Post, "Minimum spanning ellipsoids," *Proc. 16th ACM Symp. Theory of Comput.*, 1984, pp. 108–116.
33. L. Prechelt, "Proben1. A Set of Neural-Network Benchmark Problems," Univ. Karlsruhe, Germany, 1994 Available <ftp://ftp.ira.uka.de/pub/neuron/proben1.tar.gz>.
34. C. E. Priebe, D. J. Marchette, J. G. DeVinney and D. A. Socolinsky, "Classification using class cover catch digraphs," *Journal of Classification* **20** (1) (2003) 3–23.
35. X. Qiu and L. Wu, "Nearest Neighbor Discriminant Analysis," *Int. J. Pattern Recognition Artif. Intell.* **20** (8) (2006) 1245–1260.
36. D. Reilly, L. Cooper and C. Elbaum, C. "A neural model for category learning," *Bio. Cybern.* **45** (1982) 35–41.
37. S. Salzberg, "A nearest hyperrectangle learning method," *Machine Learning* **6** (1991) 277–309.
38. B. W. Silverman and D. M. Titterton, "Minimum covering ellipses," *SIAM Journal Sci. Stat. Comput.* **1** (1980) 401–409.
39. G. Toussaint, "Proximity graphs for nearest neighbor decision rules: recent progress," *Proc. 34th Symp. Comput. Stat. INTERFACE*, Montreal, Canada, 2002, pp. 83–106.

16 *H. Fayed, A. Atiya & S. Hashem*

40. N. Tsumura, K. Itoh, and Y. Ichioka, "Reliable classification by double hyperspheres in pattern vector space," *Pattern Recognition* **28** (1995) 1621–1626.
41. S. Volmer, "Fast approximate nearest-neighbor queries in metric feature spaces by buoy indexing," *Proc. 5th Int. Conf. Visual Information Systems*, Hsin Chu, Taiwan, 2002, pp. 36–49.
42. E. Welzl, "Smallest enclosing disks (balls and ellipsoids)," *Lecture Notes Comput. Sci.* 555, 1991, pp. 359–370.
43. D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE Trans. Sys. Man. Cybern.* **2**, (1972) 408–421.
44. P. A. Wolfe, "Duality theorem for nonLinear programming," *Quarterly of Applied Mathematics* **19** (1961) 239–244.
45. S. J. Wright, *Primal-dual interior-point methods*, Philadelphia: SIAM Publications, 1997.
46. H. Zhang, and G. Sun, "Optimal reference subset selection for nearest neighbor classification by tabu search," *Pattern Recognition* **35** (2002) 1481–1490.

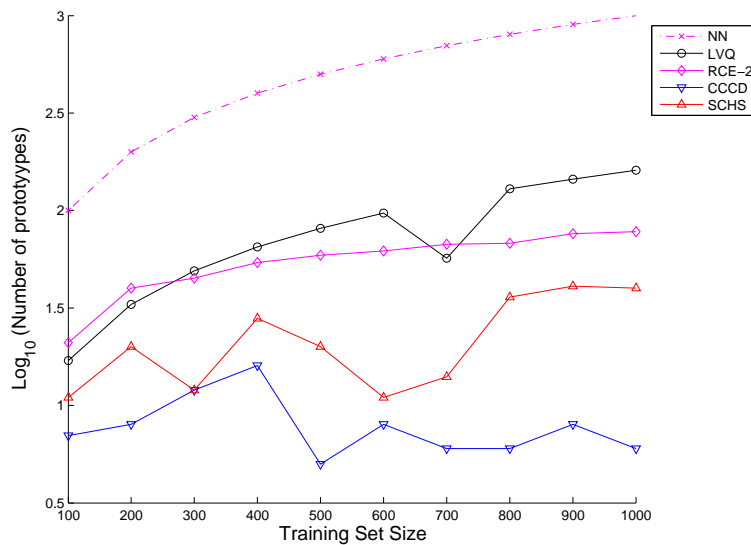


Fig. 4. Number of prototypes/hyperspheres for *PARABOLA_05* data set.

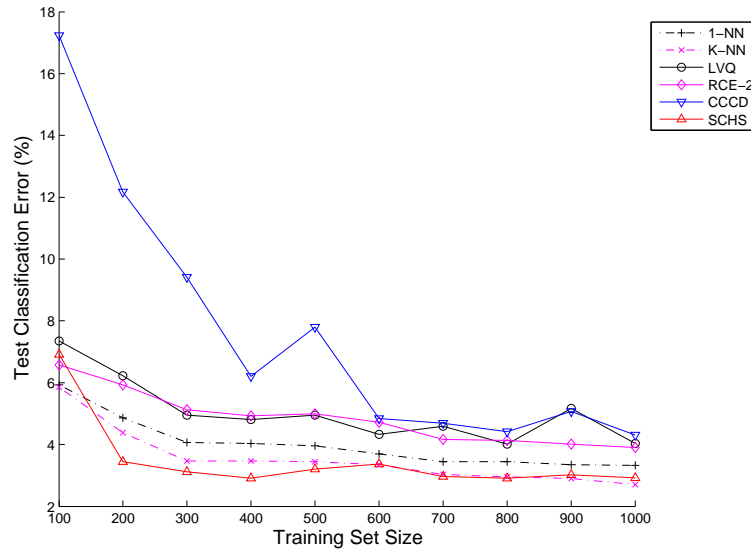


Fig. 5. Test classification error for *PARABOLA_05* data set.

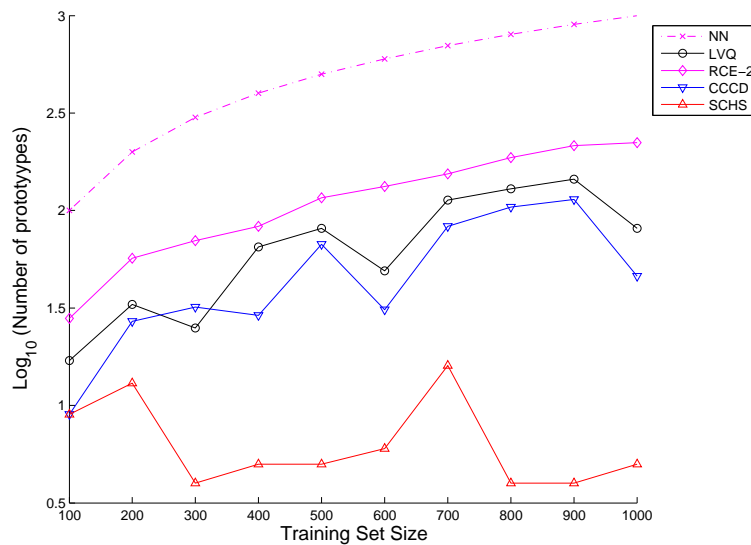


Fig. 6. Number of prototypes/hyperspheres for *DIAGONAL* data set.

18 *H. Fayed, A. Atiya & S. Hashem*

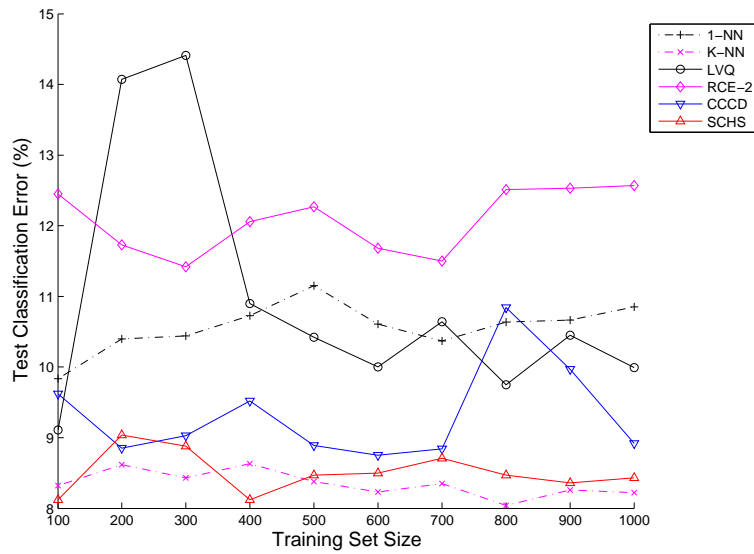


Fig. 7. Test classification error for *DIAGONAL* data set.

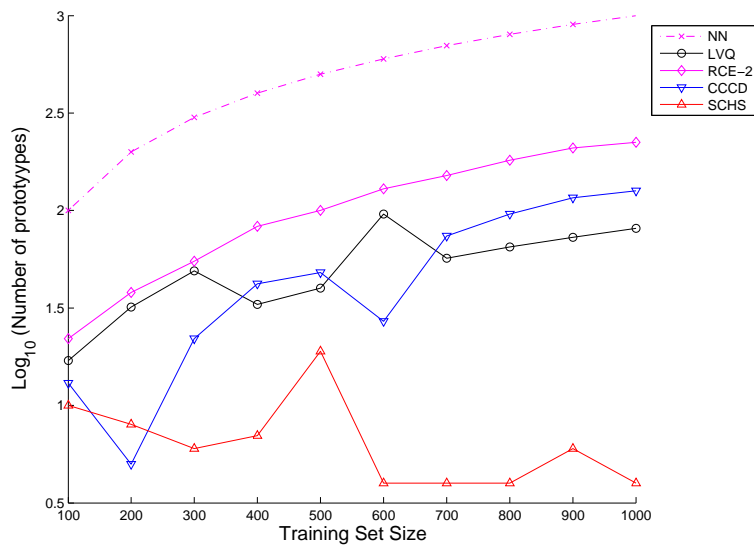


Fig. 8. Number of prototypes/hyperspheres for *INTERVAL* data set.

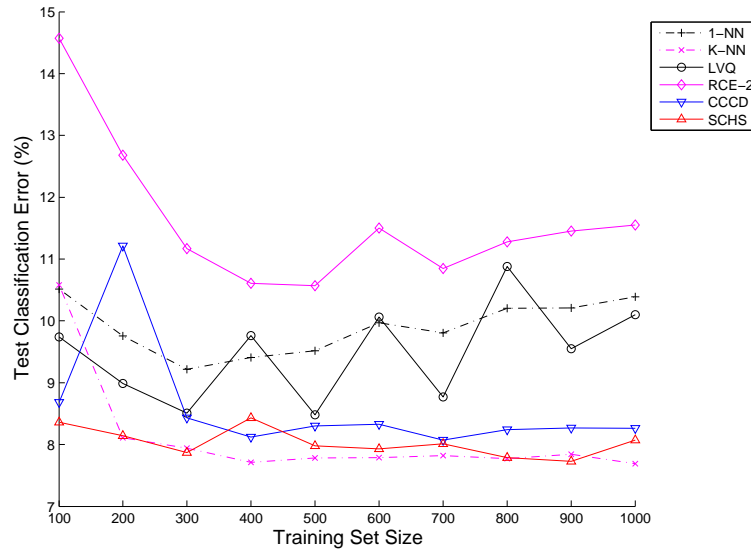


Fig. 9. Test classification error for *INTERVAL* data set.

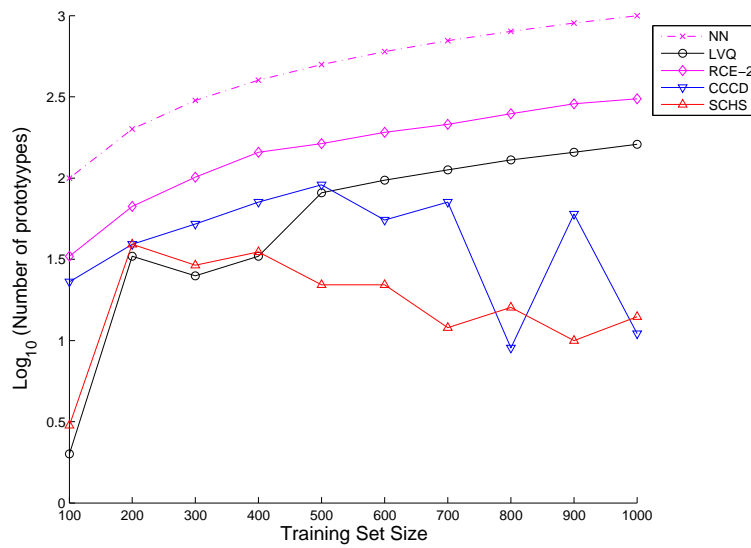


Fig. 10. Number of prototypes/hyperspheres for *I-I* data set.

20 *H. Fayed, A. Atiya & S. Hashem*

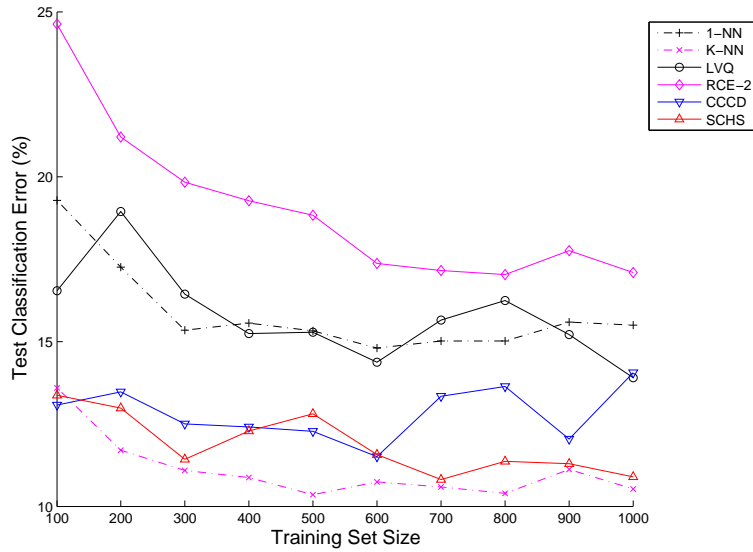


Fig. 11. Test classification error for *I-I* data set.

Table 1. Training and test times of different methods for *PARABOLA_05* data set.

Training set size	Average CPU time elapsed in training (sec.)						CPU time elapsed in testing (sec.)						
	K-NN	LVQ	RCE	RCE-2	CCCD	SCHS	1-NN	K-NN	LVQ	RCE	RCE-2	CCCD	SCHS
100	0.01	0.05	0.16	0.09	0.01	0.05	1.03	0.52	0.09	1.28	0.84	0.69	0.10
200	0.01	0.06	0.31	0.19	0.01	0.09	0.72	3.58	0.14	1.05	1.80	0.94	0.14
300	0.03	0.06	0.34	0.48	0.03	0.10	1.45	4.00	0.25	1.22	1.76	1.55	0.10
400	0.03	0.12	0.64	0.64	0.06	0.13	2.06	3.83	0.35	1.34	1.13	2.06	0.16
500	0.05	0.16	0.80	0.58	0.07	0.15	2.38	5.11	0.46	1.36	1.86	2.49	0.14
600	0.05	0.16	0.94	0.78	0.09	0.18	2.59	5.02	0.52	0.73	1.97	2.95	0.10
700	0.09	0.21	1.02	0.74	0.11	0.18	2.86	6.83	0.30	1.05	1.67	3.39	0.10
800	0.09	0.23	1.67	0.91	0.16	0.19	2.03	6.89	0.75	1.14	1.48	3.81	0.18
900	0.09	0.26	1.59	1.08	0.21	0.22	3.47	7.59	0.84	0.95	1.58	4.30	0.20
1000	0.09	0.31	1.67	1.67	0.23	0.23	3.72	7.95	0.80	1.11	1.72	4.70	0.20

Hyperspherical Prototypes for Pattern Classification 21

Table 2. Number of prototypes and test classification error of different methods for *PARABOLA_05* data set. Bold numbers indicate the minimum classification error while underlined numbers indicate the second minimum.

Training set size	Number of prototypes/hyperspheres					Test classification error (%)						
	LVQ	RCE	RCE-2	CCCD	SCHS	1-NN	K-NN	LVQ	RCE	RCE-2	CCCD	SCHS
100	17	14	21	7	11	<u>5.94</u>	5.86	7.35	12.83	6.58	17.23	6.91
200	33	22	40	8	20	4.86	<u>4.38</u>	6.23	12.16	5.93	12.17	3.45
300	49	22	45	12	12	4.07	<u>3.47</u>	4.95	11.84	5.13	9.41	3.12
400	65	23	54	16	28	4.04	<u>3.47</u>	4.81	12.60	4.93	6.20	2.91
500	81	26	59	5	20	3.95	<u>3.45</u>	4.95	12.34	4.99	7.19	3.21
600	97	25	62	8	11	<u>3.69</u>	3.35	4.33	12.35	4.72	4.84	3.37
700	57	25	67	6	14	<u>3.45</u>	3.03	4.59	12.68	4.17	4.69	2.96
800	129	26	68	6	36	<u>3.44</u>	2.95	4.01	12.41	4.13	4.42	2.91
900	145	28	76	8	41	3.35	2.90	5.17	15.49	4.01	5.07	<u>3.02</u>
1000	161	28	78	6	40	3.33	2.70	4.03	15.49	3.90	4.31	<u>2.92</u>

Table 3. Training and test times of different methods for *DIAGONAL* data set.

Training set size	Average CPU time elapsed in training (sec.)						CPU time elapsed in testing (sec.)						
	K-NN	LVQ	RCE	RCE-2	CCCD	SCHS	1-NN	K-NN	LVQ	RCE	RCE-2	CCCD	SCHS
100	0.01	0.05	0.14	0.25	0.01	1.04	0.08	3.05	0.10	0.66	0.91	0.53	0.10
200	0.01	0.06	0.45	0.36	0.02	1.91	0.13	3.03	0.16	1.47	1.86	0.94	0.10
300	0.03	0.09	0.64	0.48	0.05	2.94	0.21	4.09	0.12	1.50	1.75	1.42	0.08
400	0.03	0.12	1.06	0.53	0.07	4.16	0.31	4.25	0.38	1.53	1.61	2.03	0.08
500	0.05	0.15	1.28	0.91	0.16	6.14	0.43	6.09	0.48	1.86	2.53	2.50	0.08
600	0.06	0.18	1.38	0.75	0.14	7.08	0.49	4.89	0.25	1.44	2.30	2.94	0.08
700	0.09	0.23	1.92	1.16	0.28	8.40	0.54	5.78	0.67	1.17	2.25	3.39	0.11
800	0.09	0.27	2.63	1.44	0.37	9.54	0.63	7.09	0.76	1.30	2.01	3.84	0.08
900	0.09	0.30	3.05	1.95	0.47	10.89	0.77	7.22	0.84	1.20	2.34	4.38	0.08
1000	0.14	0.29	2.95	1.67	0.40	11.15	0.79	7.92	0.41	1.20	2.50	4.69	0.08

Table 4. Number of prototypes and test classification error of different methods for *DIAGONAL* data set.

Training set size	Number of prototypes/hyperspheres					Test classification error (%)						
	LVQ	RCE	RCE-2	CCCD	SCHS	1-NN	K-NN	LVQ	RCE	RCE-2	CCCD	SCHS
100	17	22	28	9	8	9.83	<u>8.32</u>	9.11	27.02	12.45	9.62	8.12
200	33	34	57	27	13	10.4	8.62	14.07	35.95	11.73	<u>8.85</u>	9.04
300	25	37	70	32	4	10.44	8.43	14.41	34.96	11.42	9.03	<u>8.88</u>
400	65	41	83	29	5	10.73	<u>8.63</u>	10.90	42.17	12.06	9.52	8.12
500	81	45	116	67	5	11.15	8.38	10.42	45.62	12.27	8.89	<u>8.47</u>
600	49	46	133	31	6	10.61	8.23	10.00	44.77	11.68	8.75	<u>8.50</u>
700	113	48	154	83	16	10.37	8.35	10.64	48.75	11.50	8.84	<u>8.71</u>
800	129	56	187	104	4	10.64	8.04	9.75	54.42	12.51	10.84	<u>8.47</u>
900	145	57	215	114	4	10.66	8.26	10.45	58.17	12.53	9.97	<u>8.36</u>
1000	81	57	223	46	5	10.85	8.22	9.99	58.19	12.57	8.92	<u>8.43</u>

Table 5. Training and test times of different methods for *INTERVAL data set*.

Training set size	Average CPU time elapsed in training (sec.)						CPU time elapsed in testing (sec.)						
	K-NN	LVQ	RCE	RCE-2	CCCD	SCHS	1-NN	K-NN	LVQ	RCE	RCE-2	CCCD	SCHS
100	0.01	0.05	0.27	0.11	0.01	0.06	0.91	0.55	0.09	1.05	1.28	0.55	0.14
200	0.01	0.07	0.41	0.38	0.01	0.10	0.64	3.75	0.14	1.25	1.58	0.94	0.13
300	0.03	0.10	0.81	0.41	0.04	0.16	1.24	3.39	0.27	1.30	1.65	1.50	0.12
400	0.04	0.12	0.69	0.76	0.08	0.25	2.27	5.78	0.16	1.32	1.80	2.05	0.12
500	0.04	0.15	1.17	0.78	0.12	0.33	2.25	5.19	0.22	1.35	2.01	2.52	0.18
600	0.06	0.18	1.70	1.06	0.13	0.38	2.75	6.80	0.57	1.36	2.12	2.95	0.11
700	0.08	0.19	2.14	1.14	0.26	0.46	2.97	5.75	0.33	1.39	2.19	3.39	0.10
800	0.10	0.23	2.20	1.70	0.36	0.54	3.08	6.72	0.38	1.42	2.34	3.81	0.10
900	0.12	0.25	2.55	1.47	0.47	0.64	3.55	7.44	0.42	1.49	2.41	4.34	0.11
1000	0.14	0.29	3.39	1.70	0.59	0.70	3.66	8.08	0.45	1.52	2.72	4.73	0.10

Table 6. Number of prototypes and test classification error of different methods for *INTERVAL data set*.

Training set size	Number of prototypes/hyperspheres					Test classification error (%)						
	LVQ	RCE	RCE-2	CCCD	SCHS	1-NN	K-NN	LVQ	RCE	RCE-2	CCCD	SCHS
100	17	18	22	13	10	10.51	10.58	9.74	19.51	14.57	<u>8.68</u>	8.36
200	32	27	38	5	8	9.76	8.11	<u>8.99</u>	25.38	12.68	11.21	8.14
300	49	33	55	22	6	9.21	<u>7.94</u>	8.51	26.83	11.17	8.43	7.87
400	33	42	83	42	7	9.41	7.71	9.76	33.51	10.61	<u>8.12</u>	8.43
500	40	57	100	48	19	9.51	7.78	8.48	34.45	10.57	8.30	<u>7.98</u>
600	96	66	129	27	4	9.97	7.79	10.06	43.24	11.50	8.33	<u>7.93</u>
700	57	73	151	74	4	9.81	7.82	8.77	45.40	10.85	<u>8.07</u>	<u>8.01</u>
800	65	79	181	96	4	10.21	7.77	10.88	47.09	11.28	<u>8.24</u>	7.79
900	73	87	209	116	6	10.21	<u>7.84</u>	9.55	47.56	11.45	8.27	7.73
1000	81	91	224	126	4	10.39	7.69	10.10	49.31	11.55	8.26	<u>8.07</u>

Table 7. Training and test times of different methods for *I-I data set*.

Training set size	Average CPU time elapsed in training (sec.)						CPU time elapsed in testing (sec.)						
	K-NN	LVQ	RCE	RCE-2	CCCD	SCHS	1-NN	K-NN	LVQ	RCE	RCE-2	CCCD	SCHS
100	0.01	0.06	0.31	0.19	0.01	0.11	0.89	2.74	0.07	0.95	1.86	0.59	0.08
200	0.02	0.08	0.63	0.34	0.03	0.25	0.84	2.81	0.19	1.27	2.08	0.98	0.22
300	0.03	0.10	0.98	0.81	0.06	0.47	1.67	4.45	0.14	1.81	1.55	1.36	0.20
400	0.05	0.14	1.20	0.88	0.10	0.74	2.33	5.66	0.21	2.59	2.94	2.22	0.21
500	0.05	0.16	1.39	0.99	0.17	1.10	2.58	5.84	0.47	2.03	2.49	2.70	0.19
600	0.07	0.21	2.13	1.27	0.18	1.70	2.72	6.75	0.58	1.67	1.91	3.22	0.19
700	0.09	0.25	2.72	2.03	0.27	2.41	3.34	6.38	0.73	2.23	2.50	3.67	0.18
800	0.10	0.33	2.95	2.97	0.18	2.73	3.05	7.03	0.81	2.27	2.52	4.13	0.18
900	0.13	0.31	4.02	3.05	0.36	3.35	3.78	7.53	0.90	2.28	2.86	4.64	0.17
1000	0.14	0.39	4.59	3.34	0.28	3.66	3.94	8.24	0.86	2.95	3.28	5.11	0.18

Table 8. Number of prototypes and test classification error of different methods for *I-I data set*.

Training set size	Number of prototypes/hyperspheres					Test classification error (%)						
	LVQ	RCE	RCE-2	CCCD	SCHS	1-NN	K-NN	LVQ	RCE	RCE-2	CCCD	SCHS
100	2	31	33	23	3	19.28	13.59	16.54	34.54	24.63	13.07	<u>13.37</u>
200	33	59	67	39	39	17.26	11.70	18.94	29.94	21.20	13.47	<u>12.98</u>
300	25	78	101	52	29	15.35	11.09	16.44	29.84	19.83	12.50	<u>11.43</u>
400	33	112	144	71	35	15.56	10.87	15.24	30.02	19.27	12.41	<u>12.29</u>
500	81	135	163	91	22	15.32	10.35	15.28	28.33	18.83	<u>12.27</u>	12.81
600	97	158	191	55	22	14.80	10.74	14.37	25.67	17.37	<u>11.51</u>	<u>11.57</u>
700	112	184	214	71	12	15.02	10.59	15.65	25.89	17.15	13.34	<u>10.81</u>
800	129	213	249	9	16	15.02	10.39	16.24	26.28	17.03	13.64	<u>11.36</u>
900	144	258	286	60	10	15.58	11.12	15.21	25.88	17.76	12.04	<u>11.29</u>
1000	161	274	307	11	14	15.50	10.53	13.90	25.44	17.09	14.05	<u>10.89</u>

Table 9. Training and test times of different methods for real world data sets.

Data set	Average CPU time elapsed in training (sec.)						CPU time elapsed in testing (sec.)						
	K-NN	LVQ	RCE	RCE-2	CCCD	SCHS	1-NN	K-NN	LVQ	RCE	RCE-2	CCCD	SCHS
Cancer	0.04	0.18	1.50	0.83	0.15	0.30	0.09	0.10	<0.01	0.03	0.16	0.08	<0.01
Diabetes	0.07	0.20	3.86	2.19	0.33	1.04	0.03	0.13	0.04	0.13	0.17	0.08	<0.01
Balance	0.06	0.18	1.41	0.75	0.10	0.65	0.02	0.09	0.01	0.05	0.06	0.06	<0.01
Thyroid	2.90	16.60	194.00	75.00	23.72	25.30	3.64	6.55	0.54	1.69	1.48	7.74	0.02
Satimage	2.30	62.40	1430.00	1368.00	8.90	43.70	3.83	3.84	0.69	1.69	1.66	8.14	0.59

Table 10. Number of prototypes and test classification error of different methods for real world data sets.

Data set	Number of prototypes/hyperspheres					Test classification error (%)						
	LVQ	RCE	RCE-2	CCCD	SCHS	1-NN	K-NN	LVQ	RCE	RCE-2	CCCD	SCHS
Cancer	5	73	77	37	4	<u>1.72</u>	<u>1.72</u>	0.57	4.60	3.45	0.57	0.57
Diabetes	93	268	296	210	30	30.21	<u>27.60</u>	29.69	50.52	37.50	28.65	26.56
Balance	76	176	191	81	93	17.20	<u>15.92</u>	17.83	57.96	58.60	<u>15.92</u>	14.65
Thyroid	866	1133	1028	604	16	<u>7.00</u>	6.33	7.78	17.78	9.72	<u>7.06</u>	7.28
Satimage	357	924	938	581	523	10.55	<u>10.55</u>	17.25	21.10	12.10	14.3	9.70