Proceedings of International Joint Conference on Neural Networks, Orlando, Florida, USA, August 12-17, 2007

A Nearest-Neighbor Method with Self-organizing Incremental Neural Network

Shen Furao, Osamu Hasegawa

Abstract— **We introduce a prototype-based nearest-neighbor method that is based on a self-organizing incremental neural network (SOINN). It automatically learns the number of prototypes necessary to determine the decision boundary, and it is robust to noisy training data. The experiments with artificial datasets and real-world datasets illustrate the efficiency of the proposed method.**

I. Introduction

k-nearest neighbors algorithm (kNN)[1] is very useful for some applications such as machine learning, data mining, natural language understanding, and information retrieval [2].

Salient advantages of kNN include its capability to learn from a small set of examples, add new information incrementally at run time, and model very complex target functions using a collection of less-complex approximations while requiring no optimization. On the other hand, its main disadvantage is that it is computationally intensive for large datasets.

In fact, kNN uses all training data as prototypes. We must reduce the number of prototypes to reduce the amount of required storage and thereby improve the classification speed. The problem becomes, for the training dataset $T = \{t_i \in \omega, i = 1, 2, ..., m\}$, to find a set P with M prototypes that represent T such that P is useful for classification using the nearest-neighbor rule.

For prototype-based algorithms, the question is how one knows when there are sufficient prototypes and how to prevent overfitting to training data. Nearly all methods select a fixed number of prototypes per class as an overfitting avoidance strategy [3][4]. But when the class distributions mutually differ, either in the number of patterns, the density of the patterns, or the shape of the classes, the optimal number of prototypes might also differ. The Nearest Subclass Classifier (NSC) [5] imposes the number of prototypes per class by introducing a variance-constraint parameter. The method presupposes that the features of each pattern contain the same amount of noise and do not model label noise; it is further assumed that undersampling of the classes is the same everywhere in feature space. Such assumptions might not apply to real-world tasks.

In this paper, we propose a prototype-based nearest-neighbor method that is based on a self-organizing incremental neural network (SOINN) [6], [7]. The goals of the proposed method are: (1) automatically learn the number of prototypes needed to represent every class, and, if needed, allocate a different number of prototypes for different classes with different distribution or shape; (2) learn new information without destroying old learned information, i.e., realize incremental learning; (3) reduce prototypes resulting from noise to thereby decrease misclassification, i.e., the proposed method must be robust to noisy training data; (4) delete useless prototypes during the classification process to increase the classification speed, i.e., only the prototypes used to determine the decision boundary should remain.

During the classification process, if not stated differently, for all experiments in this paper we will employ the 1-NN (prototype) rule to classify patterns based on the generated set of labeled prototypes.

Shen Furao is with the State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing, 210093, China (email: frshen@nju.edu.cn) and Japan Society for the Promotion of Science; Osamu Hasegawa is with the Imaging Science and Engineering Laboratory, Tokyo Institute of Technology.

II. Overview of self-organizing incremental neural network (SOINN)

In this section, we introduce the self-organizing incremental neural network (SOINN) [6][7], which is the basis of the proposed method. Fundamentally, SOINN adopts a two-layer network. The training results of the first layer are used as the training set for the second layer. The goals of SOINN are realizing unsupervised learning and representing the topology of the input distribution.

To insert new nodes and thereby realize incremental learning and topology representation, SOINN adopts two schemes: between-class insertion and within-class insertion. In between-class insertion, SOINN uses a threshold distance T to judge whether two samples belong to the same cluster. The sample will be inserted to the network as a new node if a sample belongs to a new cluster. In within-class insertion, SOINN uses a local-error-based insertion criterion by which, on a time average, nodes with high error serve as criteria to insert a new node. To realize within-class insertion, SOINN uses five user-determined parameters and another parameter, 'error-radius', to judge whether the insertion is successful. The latter feature complicates the system and makes it difficult to understand.

The similarity threshold T_i is defined as the distance (Euclidean) from the boundary to the center of the Voronoi region V_i of node i. During the learning process, node i will change its position to meet the inputting pattern distribution. Consequently, the Voronoi region V_i of the node i will also change; for that reason, the similarity threshold T_i will also change.

To build connections between neural nodes, SOINN adopts the competitive Hebbian rule [8]: for each input signal,

1-4244-1380-X/07/$25.00 ©2007 IEEE

connect the two closest nodes (*winner* and *second winner*, measured by Euclidean distance) by an edge; remove connections that have not been refreshed for a while. Here, *winner* designates the nearest node to the input pattern, and *second winner* denotes the second-nearest node to the input pattern.

To delete the nodes caused by noise, SOINN removes those nodes with no or only one topological neighbor if the number of input signals generated so far is an integer multiple of a parameter λ. With the help of second-layer network, SOINN can delete a slightly higher density of noise and even separate the classes with low-density overlap.

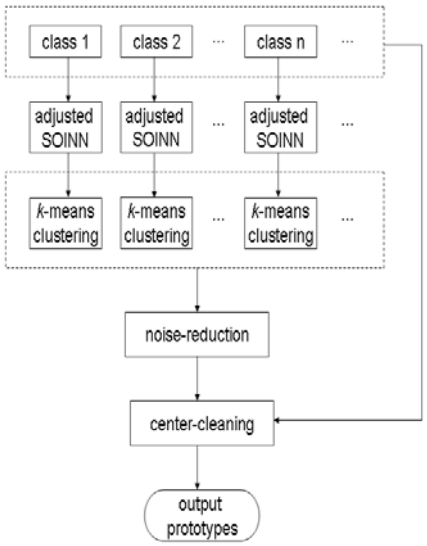

Fig. 1: Learning process of Adjusted SOINN Classifier

III. The Proposed Classifier

A shortcoming of SOINN is that it has too many parameters to realize within-class insertion. Another is that it is not stable: its results depend on the input sequence of training data. Moreover, the target of SOINN is to realize unsupervised learning and topology representation. In this study, we seek to perform supervised learning and use as few prototypes as possible to realize rapid classification. We improve SOINN in the following respects: (1) adjust SOINN with fewer parameters to represent the topological structure of input data; (2) improve SOINN to obtain more stable results using k-means clustering [3]; (3) design the noise-reduction process to reduce some prototypes resulting from noise; and (4) design a center-cleaning process to delete those prototypes that are useless for the classification process. The proposed method is based on SOINN. Therefore, we named the proposed algorithm: Adjusted SOINN Classifier (ASC). ASC inherited some properties of SOINN such as incremental learning, robustness against noise, and automatic learning of numerous prototypes necessary to represent every class. A flowchart of ASC is shown in Fig. 1.

Figure 1 illustrates that ASC performs adjusted-SOINN and k-means clustering separately for every class. Then it uses all k-means results to perform noise-reduction. Finally, ASC uses the input data of all classes and results of noise-reduction processes to perform center-cleaning processes.

A. Adjusted SOINN

During training of the first layer of SOINN, between-class insertion is the main part; within-class insertion contributes little for inserting new nodes. The target of the second layer is to delete redundant nodes, separate overlapped clusters, and delete nodes resulting from noise. For the topology-representation target, the first layer can achieve better topology representation than the second layer. Here we only adopt the first layer of SOINN as the basis of the proposed ASC method, then delete the within-class insertion part to ease its comprehension and retain five user-determined parameters. The deletion of within-class insertion will not influence the learning results because, if we only adopt a single-layer network, the between-class insertion assures that the node density will be sufficient to represent the topology structure. With the definition of the similarity threshold, newly inserted nodes might come from the not happened area, and the subsequent processes (e.g., competitive Hebbian rule) will link such new nodes with old nodes. The adaptively updated similarity threshold will not be too large to make the nodes sparse. *Algorithm 3.1* is the detailed algorithm of adjusted SOINN. W_i is the weight vector of node i.

Algorithm 3.1: Adjusted SOINN

1) Initialize node set A to contain two nodes, c_1 and c_2 with weight vectors chosen randomly from the input pattern. Initialize connection set C, $C \subset A \times A$, to the empty set.
2) Input new pattern $\xi \in R^n$.
3) Search the nearest node (*winner*) s_1, and the second-nearest node (*second winner*) s_2 by $s_1 = \arg\min_{c \in A} \|\xi - W_c\|$, $s_2 = \arg\min_{c \in A \setminus \{s_1\}} \|\xi - W_c\|$. If the distance between ξ and s_1 or s_2 is greater than similarity threshold T_{s_1} or T_{s_2}, the input signal is a new node, add it to A and go to Step2) to process the next signal.
4) If a connection between s_1 and s_2 does not exist, create it. Set the age of the connection between s_1 and s_2 to zero.
5) Increment the age of all edges emanating from s_1 by 1.
6) Adapt the weight vectors of the *winner* and its direct topological neighbors by fraction $\epsilon_1(t)$ and $\epsilon_2(t)$ of the total distance to the input signal,
   $\Delta W_{s_1} = \epsilon_1(t)(\xi - W_{s_1})$
   $\Delta W_i = \epsilon_2(t)(\xi - W_i)$ for all direct neighbors i of s_1
   We adopt a scheme to adapt the learning rate over time by $\epsilon_1(t) = 1/t$ and $\epsilon_2(t) = 1/100t$.
7) Remove edges with an age greater than a predefined threshold a_d. If this results in nodes having no more emanating edges, remove them as well.
8) If the number of input signals generated so far is an integer multiple of parameter λ, delete some nodes as

follows: for all nodes in A, search for nodes having no neighbor or only one neighbor, then remove them.

9) Go to Step2 to continue the learning until the learning time is satisfied.

For *Algorithm 3.1*, we must determine two parameters a_d and λ. These two parameters will influence the frequency of deletion of the connections between nodes and nodes located in sparse areas. If we wish to retain previous learned knowledge much longer, we choose a large value for these two parameters, and obtain many nodes to realize a high recognition ratio. If we desire fewer nodes to save memory space and hasten the classification, we set the respective values of these two parameters small to remove nodes and edges frequently. The two parameters are therefore dependent upon the real condition of the task; we can use these parameters to control the recognition performance of ASC.

B. k-means clustering

An often-used method for finding clusters and cluster centers in a set of unlabeled data is k-means clustering [3]. Given the desired number of cluster centers m, we seek an initial set of centers $c_j(0), j = 1, \ldots, m$. The k-means algorithm alternates the following two steps: (1) For each center, we identify the subset of training points (its cluster) that is closer to it than any other center. (2) The means of each feature for the data points in each cluster are computed; this mean vector becomes the new center for that cluster. Those two steps are iterated until convergence.

In fact, k-means depends heavily on the initial value of centers $c_j(0), j = 1, \ldots, m$. The difficulties of k-means are determination of the number of centers m in advance and finding good initial values of centers. To resolve such difficulties, in ASC, we use the learned number of nodes of adjusted SOINN as the number of centers, and use the position (weight vector) of adjusted SOINN nodes as the initial value of centers.

In ASC, adjusted SOINN is not stable: the results depend on the sequence of the input data. The number and the position of nodes are different if we repeat the training under the same environment with a different input sequence. Using k-means clustering, we move such nodes to the center of the clusters and thereby improve the ASC stability. Because that the generated nodes of adjusted SOINN represent the topology of input data, such nodes are distributed near the centers of sub-clusters; k-means clustering moves such nodes to the centers of sub-clusters.

C. Noise-reduction process

If the training data include noise, some nodes will be generated by noise during adjusted SOINN and k-means clustering. Because we only adopt a single layer in adjusted SOINN, if much noise exists, we cannot remove all noise-generated nodes. To prevent the nearest-neighbor rule from fitting the noisy training data without restriction, we use the idea of an early method k-Edited Neighbors Classifier (ENC) [9] in ASC: if the node label differs from the label of majority voting of its k-neighbors, it is considered an outlier and the node is removed from the set of prototypes.

In this noise-reduction process, we can use some methods such as cross-validation to tune parameter k. In fact, during the test of real-world data in Section 4, we perform no noise-reduction because we presume that noise is very sparse among real-world data. Furthermore, such noise is deleted during the adjusted SOINN training. In this situation, the usage of noise-reduction process might even delete some nodes that are important for determining the decision boundary. However, for artificial data, we use high-density overlapped classes and add 10% noise to the dataset, the usage of noise-reduction processes can improve the recognition performance. We will undertake that discussion in experiment section 4.

D. Center-cleaning process

The prototype set obtained using adjusted SOINN and k-means clustering is useful to represent the topological structure of input data. However, during the classification process, some prototypes in the central part of a class might be useless: during the classification process, we use the 1-NN (prototype) rule to classify patterns based on the generated set of labeled prototypes. For that reason, only prototypes that lie in the boundary can be used. We must delete those prototypes that lie in the central parts of classes to conserve memory space of storage for prototypes and thereby increase the classification rate. We designed *Algorithm 3.2* to realize that goal. *Algorithm 3.2* is based on this idea: if a prototype of a class has never been the nearest prototype to other classes, the prototype lies in the central part of the class, and we delete it.

Algorithm 3.2: Center-cleaning process

1) Presume there are n classes, given class $i, i = 1, \ldots, n$. Then perform the following steps.
2) For all samples of other classes that differ from class i, find the nearest prototype of class i generated by adjusted SOINN and k-means clustering.
3) If a prototype of class i has never been the nearest prototype of other classes, delete the prototype. Repeat this step until no prototype can be deleted.

For the prototype-based classifier, the recognition ratio, storage requirements, and recognition speed are useful to evaluate the classifier performance. For this study, we measure the memory requirements and the increase of classification speed with the compression ratio r_c of the trained classifiers, where $r_c = \frac{number\ of\ prototypes}{train\ data\ size}$, and use the correct recognition ratio and compression ratio to validate the performance of classifiers.

IV. Experiment

In this part, we first use some two-dimensional (2D) artificial data sets to test the ASC and illustrate the detail of ASC. Then, through a test performed on a real-word dataset, we prove the efficiency of the proposed method. Finally, we compare ASC with some other typical prototype-based classifiers.

A. Artificial dataset

During the test of all artificial datasets, we set the parameters as follows: $a_d = 20$, $\lambda = 20$, $k = 5$. The a_d and λ will influence the number of generated prototypes, i.e. the compression ratio, but they will not influence the recognition ratio if we can obtain sufficient prototypes. The parameter k is not sensitive: we can choose another value for it and obtain the same recognition results. For all experiments in this section, we do 10-times learning and testing, and then give the average recognition ratio and compression ratio as the recognition performance.

1) Two Gaussian distribution without overlap
   In this experiment, we adopt two non-overlapped Gaussian distributions (as shown in Fig. 2). For every class, we choose 2,000 samples as the training pattern, and choose 200 samples as the test pattern. There are 4,000 training samples and 400 test samples. The correct recognition ratio of the 1-NN classifier for this example is 100%.
   Then we train the ASC. Fig. 3 depicts the adjusted SOINN result, showing that adjusted SOINN can represent the topology of two classes well. Fig. 4 shows the training results of ASC. To realize the classification, ASC requires only six prototypes (three for one class, and three for the other class). Using these six prototypes, we classify the test set and obtain a 100% correct recognition ratio. Compared to the 1-NN classifier, ASC uses 6/4000 = 0.15% (compression ratio $r_c = 0.15\%$) prototypes of 1-NN and obtains the same recognition ratio.

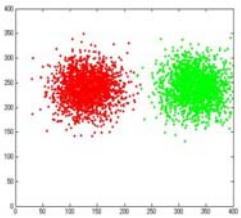

Fig. 2: Two Gaussian distributions data sets without overlap (2 classes)

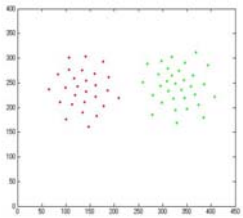

Fig. 3: Adjusted SOINN results of Fig. 2

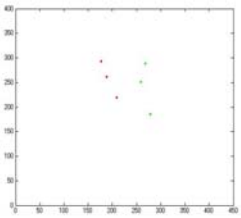

Fig. 4: ASC results of Fig. 2

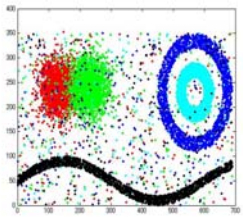

Fig. 5: Five classes with overlap and noise

2) Five classes with different distribution and different shape, with overlap and 10% noise
   In this experiment, we add three other classes to the classes in Experiment 1. The classes obey the distribution shown in Fig. 5. They are two Gaussian distributions, two concentric rings, and one sinusoid curve. We move two Gaussian distribution classes together to form overlapped classes, and add 10% noise to the classes. The noise is distributed randomly on the whole feature space and we randomly label such noise samples with class names. Fig. 5 is the distribution, which shows that some samples belong to one class, but that they have different class labels. We randomly selected 2,000 from each class as the training pattern, and randomly selected 200 samples from every class as the test pattern: consequently, there are 10,000 samples in the training set and 1,000 samples in the test set. In the training samples, there are noise samples: some samples belong to one class but are labelled as being of different classes. We use the test set without noise to test how the noise in the training data influences the recognition performance. The correct recognition ratio of 1-NN is 94.1%.
   Fig. 6 shows the adjusted SOINN results. Even adjusted SOINN apparently has some noise deletion function, but some prototypes resulting from noise remain undeleted. Fig. 7 shows ASC results: the noise-reduction process deleted those noise prototypes. The correct recognition ratio of ASC is 97.8% and it requires 87 prototypes: ASC uses 87/10,000 = 0.87% ($r_c = 0.87\%$) prototypes of 1-NN and obtains a higher recognition ratio. This experiment suggests that ASC is robust to noise because, even though 10% noise exists, ASC can achieve a satisfy recognition ratio with high compression ratio.

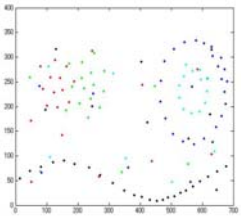

Fig. 6: Adjusted SOINN results of Fig. 5

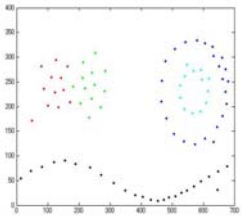

Fig. 7: ASC results of Fig. 5

B. Real world data

As described in Section III-C, for real-world data, we presume that there are very sparse noise samples and adjusted SOINN can remove those nodes caused by noise. Consequently, we delete the noise-reduction process from ASC for all experiments in this section.

In this experiment, we use Optical Recognition of the Handwritten Digits database (Optdigits) [10] to test ASC. This dataset includes 10 classes (handwritten digits) from 43 people: 30 contributed to the training set and a different 13 to the test set. In all, 3823 samples are included in the training set; 1797 samples are in the test set. The dimension of the samples is 64. The correct recognition ratio of 1-NN is 98%.

During the ASC training, we tested three different parameter sets for (a_d, λ): (1) (50, 50), (2) (25, 25), and (3) (10, 10). We performed 10 training iterations for every parameter set, and took the average results as the final results. Table 1 lists the number of prototypes of every class for different

parameter sets. Using such prototypes, we classify the test samples to different classes and show recognition results for different parameter sets in Table 2. Table 1 shows that, for different classes, the necessary number of prototypes is different; larger $\{a_d, \lambda\}$ will engender more prototypes. Table 2 shows that, (1) with the correct recognition ratio 97.7% (Column II of Table 2), ASC speeds up the traditional 1-NN method by more than 10 times and uses less than 10% of the memory space ($r_c = 9.9\%$); (2) if we want to speed up the classification process using much less memory space, the correct recognition ratio must be decreased (Column II – Column IV of Table 2); also, the parameter sets (a_d, λ) can be used to control the recognition performance.

Table 1: The number of prototypes, displayed with the average and standard deviation

Class	No. of nodes for different sets of (a_d, λ)		
	(50, 50)	(25, 25)	(10, 10)
0	32 ± 4	20 ± 3	11 ± 3
1	40 ± 4	31 ± 4	11 ± 1
2	38 ± 3	27 ± 2	12 ± 5
3	41 ± 5	24 ± 3	12 ± 3
4	35 ± 3	25 ± 2	9 ± 3
5	39 ± 5	28 ± 5	10 ± 3
6	34 ± 3	25 ± 2	11 ± 3
7	33 ± 3	24 ± 2	12 ± 2
8	39 ± 5	25 ± 4	12 ± 3
9	45 ± 5	30 ± 3	12 ± 2
Total number	377 ± 12	258 ± 7	112 ± 7

For reference, we also compared ASC with a support vector machine (SVM) using the Optdigits dataset. Passerini et al. calculated the recognition ratio of multiclass SVM with different kernel functions [11]: for One-vs.-All SVM methods, the best recognition ratio of Optdigits is 97.2%; for All-pairs SVM methods, the best recognition ratio of Optdigits is 97.4%. Table 2 shows that the ASC can attain equivalent or better recognition results than those of SVM.

Table 2: Recognition performance of ASC, displayed with average and standard deviation

	Parameter set of $\{a_d, \lambda\}$		
	(50, 50)	(25, 25)	(10, 10)
recognition ratio (%)	97.7 ± 0.2	97.4 ± 0.2	97.0 ± 0.2
No. of prototype	377 ± 12	258 ± 7	112 ± 7
Compression ratio (%)	9.9 ± 0.3	6.8 ± 0.2	2.9 ± 0.2

Veenman and Reinders compare the Nearest Subclass Classifier (NSC; σ) [5] to some other classifiers including the K-Means Classifier (KMC; M) [3], the k-Nearest Neighbors Classifier (NNC, k) [1], the k-Edited Neighbors Classifier (ENC; k) [9], the Bootstrap technique (BST; M) with T=100 and $k = 3$ [4], and Learning Vector Quantization (LVQ; M) with $\alpha = 0.3$, $\eta = 0.8$, and T=100 as in [4]. For the methods described above, Veenman and Reinders optimize the parameter using tuning by cross-validation. They also compare the NSC classifier with some methods with fixed parameters such as NNC (3), RT3 (3), and TAB (α) with $\alpha = 0.05$, $T_t = 0.03N$, $T_i = 20$, and $T = 100$ as in [4]. They show that the tuning of parameters can greatly improve the recognition performance. In summary, Veenman and Reinders assert that the NSC classifier works better than the above-mentioned classifiers from the balance of recognition ratio and classification speed.

Table 3: Datasets used for comparison

Data set	objects	features	classes
Iris	150	2	2
Breast cancer	683	9	2
Ionosphere	351	34	2
Glass	214	9	6
Liver disorders	345	6	2
Pima Indians	768	8	2
Wine	178	13	3

In this section, we compare the proposed ASC with some other prototype-based classifiers to evaluate the ASC. Furthermore, we use the results in **TABLE 2** of [5] to compare the proposed ASC with NSC, KMC, and LVQ. The recognition performance of some other classifiers is described elsewhere [5].

In [5], the authors use nine datasets to evaluate NSC and compare it with other methods. These nine datasets include one artificial dataset produced by the authors, and a Sonar dataset with 30 feature vectors. It is difficult to regenerate the artificial data set and the 30 feature Sonar dataset (in general, the Sonar dataset has 60 feature vectors). Therefore, we compare the proposed ASC with NSC and other classifiers with seven other datasets. We list such datasets in Table 3. Details of the datasets are described elsewhere [10].

In ASC without the noise-reduction process, two parameters cannot be learned directly from the training data, we use a general ten-fold cross-validation to tune such parameters, those results are listed in Table 4(b).

Table 4 lists comparison results of ASC and other classifiers. Table 4(a) is the correct recognition ratio. For Iris, Breast cancer, Glass, and Wine datasets, ASC provides the highest recognition ratio, and for Ionosphere and Pima Indians datasets, the recognition ratio is near the best classifier. The last rows in Table 4(a) show the average recognition ratio of all datasets and summarizes the performance of all algorithms: compared with other prototype-based classifiers, ASC achieves the best recognition ratio. Table 4(b) shows the compression ratio: for all datasets, ASC provides the best or nearly best compression ratio. Overall, ASC achieves the best compression ratio, meaning that ASC requires less memory space and processing time than other classifiers. The bold-typeface data in Table 4 is the best or near-best classifier.

Table 4 also shows that some prototype-based classifiers such as NNC and NSC can obtain a high recognition ratio, but they are time-consuming. For some rapid classifiers such as KMC and LVQ, the recognition ratios are low. The proposed ASC attains the highest recognition ratio, in addition to the highest classification speed. Compared to the recently published NSC method, on average, ASC attains an 82.3% recognition ratio with a 4.6% compression ratio, whereas NSC requires a 34.2% compression ratio to attain

a 80.4% recognition ratio. We infer that ASC is much more efficient than NSC.

Table 4: Comparison of results from ASC and other classifiers

Data set	ASC (a_d, λ)	NSC (σ^2_{max})	KMC (M)	NNC (k)	LVQ (M)
Iris	**97.4** ± 0.86	96.3 ± 0.4	96.2± 0.8	96.7± 0.6	96.1± 0.6
Breast cancer	**97.4** ± 0.38	**97.2** ± 0.2	95.9± 0.3	**97.0**± 0.2	96.3± 0.4
Ionos phere	**90.4** ± 0.64	**91.9** ± 0.8	87.4± 0.6	86.1± 0.7	86.4± 0.8
Glass	**73.5** ± 1.6	70.2 ± 1.5	68.8± 1.1	72.3± 1.2	68.3± 2.0
Liver disor-ders	62.6 ± 0.83	62.9 ± 2.3	59.3± 2.3	**67.3**± 1.6	66.3± 1.9
Pima Indians	72.0 ± 0.63	68.6 ± 1.6	68.7± 0.9	**74.7**± 0.7	73.5± 0.9
Wine	**82.6** ± 1.55	75.3 ± 1.7	71.9± 1.9	73.9± 1.9	72.3± 1.5
Average	**82.3** ± 0.93	80.4 ± 1.2	78.3± 1.1	81.1± 0.99	79.9± 1.2

(a) Recognition ratio, displayed with average and standard deviation

Data set	ASC (a^*_d, λ^*)	NSC $(\sigma^{2\ *}_{max})$	KMC (M^*)	NNC (k^*)	LVQ (M^*)
Iris	5.2 (6, 6)	7.3 (0.25)	8.0 (4)	100 (14)	15 (22)
Breast cancer	**1.4** (8, 8)	1.8 (35.0)	0.29 (1)	100 (5)	5.9 (40)
Ionos phere	**3.4** (15, 15)	31 (1.25)	4.0 (7)	100 (2)	6.8 (24)
Glass	**13.7** (15, 15)	97 (0.005)	17 (6)	100 (1)	45 (97)
Liver disor-ders	**4.6** (6, 6)	**4.9** (600)	11 (19)	100 (14)	8.4 (29)
Pima Indians	**0.6** (6, 6)	1.7 (2600)	1.0 (4)	100 (17)	3.4 (26)
Wine	**3.2** (6, 6)	96 (4.0)	29 (17)	100 (1)	32 (57)
Average	**4.6**	34.2	10.0	100	16.6

(b) Compression ratios. The optimal parameter value tuned by cross-validation is shown in ( ).

According to [5], the tuning of parameters requires much time for the NSC; the same is true for some other methods. It is also noteworthy that the ASC training is more rapid than for some classifiers such as NSC, the tuning of parameters for ASC is not time consuming, it is only slightly slower than KMC (k). The tuning speed is much higher than that of NSC.

V. Conclusion

In this paper, we proposed a new prototype-based classifier based on an adjusted self-organizing incremental neural network (SOINN). We call this method the adjusted-SOINN classifier (ASC). Using an adaptive similarity threshold, the system can grow incrementally and accommodate input patterns of incremental data distribution. By deleting within-class insertion, the system requires fewer parameters than SOINN. The use of ASC can reduce the prototypes resulting from noise and render the method robust to noise and possibly obtain a high recognition ratio. Deletion of useless prototypes during classification makes ASC much faster than some other classifiers. Even in cases where it is necessary to tune the parameters through cross-validation, compared to other classifiers, the tuning is not time-consuming. Compared to other fast nearest-neighbor classifiers, the training of ASC is also much faster. In the experiment, ASC is compared to some other classifiers. The recognition ratio and compression ratio show that ASC obtains superior performance and shows itself as a very efficient method.

Acknowledgment

Funding for this study was primarily provided by New Energy and Industrial Technology Development Organization. The authors express their appreciation.

This study was primarily supported by Industrial Technology Research Grant Programin '04 from New Energy and Industrial Technology Development Organization (NEDO) of Japan. This work was supported in part by the National Natural Science Foundation of China under grant no. 60573157.

References

[1] T.M. Cover, P.E. Hart, "Nearest Neighbor Pattern Classifictaion," IEEE Trans. on Information Theory, Vol. IT-13, No. 1, pp. 21-27, 1967.

[2] B.V. Dasarathy, "Nearest Neighbor(NN) Norms: NN Pattern Classification Techniques," Los Alamitos, Calif.: IEEE CS Press, 1991.

[3] T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, 2001.

[4] J.C. Bezdek, L.I. Kuncheva, "Nearest Prototype Classifier Design: An Experimental Study," Int'l J. Intelligent Systems, Vol. 16, pp. 1445-1473, 2001.

[5] C.J. Veenman and M.J.T. Reinders, "The Nearest Subclass Classifier: A Compromise between the Nearest Mean and Nearest Neighbor Classifier," IEEE Tracs. Pattern Analysis and Machine Intelligence, Vol. 27, No. 9, pp. 1417-1429, 2005.

[6] F. Shen and O. Hasegawa, "An on-line learning mechanism for unsupervised classification and topology representation," IEEE Computer Society International Conference on Computer Vision and Pattern Recognition (CVPR 2005), San Diego, CA, USA, June 20-26, 2005.

[7] F. Shen and O. Hasegawa, "An Incremental Network for On-line Unsupervised Classification and Topology Learning," Neural Networks, Vol. 19, pp. 90-106, 2006.

[8] T.M. Martinetz, "Competitive Hebbian Learning Rule Forms Perfectly Topology Preserving Maps," ICANN, pp. 427-434, 1993.

[9] D.L. Wilson, "Assymptotic Properties of Nearest Neighbor Rules Using Edited Data," IEEE Trans. Systems, Man, and Cybernetics, Vol. 2, No. 3, pp. 408-421, 1972.

[10] C. Merz, M. Murphy, "UCI Repository of Machine Learning Databases, Irvine, CA," University of California Department of Information, 1996. [http://www.ics.uci.edu/mlearn/MLRepository.html]

[11] A. Passerini, M. Pontil, and P. Frasconi, "From Margins to Probabilities in Multiclass Learning Problems," Proceedings of the 15th European Conference on Artificial Intelligence, 2002.