Neural Comput & Applic (2012) 21:1205–1215
DOI 10.1007/s00521-010-0511-4

ORIGINAL ARTICLE

An incremental learning vector quantization algorithm for pattern classification

Ye Xu · Furao Shen · Jinxi Zhao

Received: 31 August 2010 / Accepted: 16 December 2010 / Published online: 4 January 2011

Abstract Prototype classifiers have been studied for many years. However, few methods can realize incremental learning. On the other hand, most prototype classifiers need users to predetermine the number of prototypes; an improper prototype number might undermine the classification performance. To deal with these issues, in the paper we propose an online supervised algorithm named Incremental Learning Vector Quantization (ILVQ) for classification tasks. The proposed method has three contributions. (1) By designing an insertion policy, ILVQ incrementally learns new prototypes, including both between-class incremental learning and within-class incremental learning. (2) By employing an adaptive threshold scheme, ILVQ automatically learns the number of prototypes needed for each class dynamically according to the distribution of training data. Therefore, unlike most current prototype classifiers, ILVQ needs no prior knowledge of the number of prototypes or their initial value. (3) A technique for removing useless prototypes is used to eliminate noise interrupted into the input data. Results of experiments show that the proposed ILVQ can accommodate the incremental data environment and provide good recognition performance and storage efficiency.

Y. Xu (✉) · F. Shen · J. Zhao
National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
e-mail: xuye.nju@gmail.com

F. Shen
e-mail: frshen@nju.edu.cn

J. Zhao
e-mail: jxzhao@nju.edu.cn

Keywords Learning vector quantization · Incremental learning · Adaptive threshold · Classification compression ratio

1 Introduction

Prototype based method [11] is a class of powerful tools in classification family. It searches for a set of representative prototypes that reflects the distribution in original data space. Then the label of a test sample, usually called query, is defined by the previously generated set of prototypes [6] shows that the error rate of prototype based classifiers is at most twice of Bayes error rate. Therefore, Prototype based method has been widely used in object classification [31], visualization recognition system [32, 30], optimization [12], and subspace learning [26].

Nearest Neighbor Classifier (NNC) [4] is the most famous prototype classifier. It applies all samples in training set as prototypes. NNC works well on large training set. But a trivial consequence of NNC is the large size of prototype sets that aggravates the computation burden. What's worse, the performance of NNC decreases rapidly if some noisy samples are interrupted in the training set. According to this, some different ways [7, 10, 13, 18, 20, 23, 25, 28] of learning prototypes have been proposed recently.

But all these typical prototype classifiers have some shortcomings. (1) Almost all of the reported prototype classifiers can not accommodate an incremental dataset. The learning effect will be very poor when it is used for incremental applications. (2) A user must initialize the number and value of the prototypes; the number of prototypes for each class is predetermined arbitrarily. Under many circumstances, we are not able to avail the prior

knowledge about the distributions of data from each class. Therefore, the learning performances will be affected by improper initial values. More importantly, online learning can not be achieved because many people often initialize them with partial training samples. (3) Many prototype classifiers can not cope with the noises that are interrupted into the training patterns. Thus it is very likely for such methods to learn a prototype from noise data.

As described herein, we propose a supervised method named Incremental Learning Vector Quantization (ILVQ) to cope with the shortcomings of those typical prototype classifiers. The primary contribution of ILVQ is to realize an incremental learning task in the meanings of within-class incremental learning and between-class incremental learning. Within-class incremental learning means that the prototype set learns new knowledge incrementally within the same class, whereas between-class incremental learning means that it incrementally learns new knowledge that comes from a new incoming class. In other words, ILVQ not only incrementally learns prototypes within a class (within-class incremental learning); it also incrementally learns the number of classes during training (between-class incremental learning). Another contribution is that ILVQ needs not users to predetermine the number or value of prototypes before learning. The proposed ILVQ can grow dynamically, learning the number of prototypes for each class needed to solve a current issue and adjusting the number and value of prototypes during training. The third contribution of ILVQ is to eliminate the noise from input data. Consequently, the noise or outliers among input patterns in the training set will have little influence on the ILVQ performance.

We organize the rest of this paper as follows. In Sect. 2, we briefly introduce the related works. ILVQ is proposed in Sect. 3. In Sect. 4, we do some experiments to compare ILVQ with other prototype classifiers to show the effectiveness of our approach. That section also highlights that ILVQ can be used in incremental learning. Finally in Sect. 5, we conclude the paper.

2 Related work

Prototype classifier relies on a set of appropriately chosen prototypes. How to choose prototypes is the key issue for prototype classifiers. According to it, prototype classifiers can be categorized into two classes. One class of prototype methods is classifier based on condensing scheme [5]. Condensing scheme focuses on finding representative prototypes that reflect the distribution of original samples. This scheme improves the generalization capacity and storage efficiency of classifiers. A simple reported condensing scheme method is K-Means Classifier (KMC) [10]. It is based on k-means prototypes positioning with 1-NN rule. Learning Vector Quantization (LVQ) [13], along with some LVQ family methods such as Generalized Learning Vector Quantization (GLVQ) [23], takes advantage of class information to move the prototype vectors, in order to improve the quality of the classifier decision region. Mollineda et al. [18] adopt geometric property and clusters to search a condensed set of prototypes. Pekalska and Duin [20] extract prototypes from proximity-based representation space.

Another class of methods is classifier based on editing scheme. Editing scheme aims at removing useless prototypes that are likely to be outliers and detecting prototypes located in the overlap among classes. This scheme is good at eliminating prototypes in low probability density regions. Thus it handles noisy data environment effectively and avoids overfitting. Some editing scheme based classifiers have been proposed. For instance, Wilson editing classifier [28] removes all examples that have been misclassified by 1-NN rule from training sets. Eick et al. [7] propose a editing scheme using supervised clusters. Nearest Subclass Classifier (NSC) [25] applies a technique to merge prototypes based on a defined variance constraint parameter.

Our work also relates to incremental learning classifiers [21] is the first reported incremental learning algorithm which incrementally train an ensemble of classifiers. During the pass few years, incremental learning classifiers have attracted many attentions [3] and been put into wide applications [14, 22]. However, to the best of our knowledge, few prototype classifiers can fulfill incremental learning. Therefore, it is desirable to design a prototype classifier to fulfill incremental learning task.

3 Incremental learning vector quantization

As mentioned in Sect. 1, we try to propose a method to cope with the shortcomings of those typical prototype classifiers. Here, we summarize the targets of the proposed ILVQ as follows:

- To realize incremental learning task including within-class incremental learning and between-class incremental learning.
- To fulfill supervised classification task without using prior knowledge such as the total number of prototypes, the amount of prototypes for each class, and the initial values of them.
- To eliminate the negative effect imposed by unreasonable initialization and probable noise in training set.

To fulfill the goals above, we propose the ILVQ as follows. The overall learning process is shown in Fig. 1.

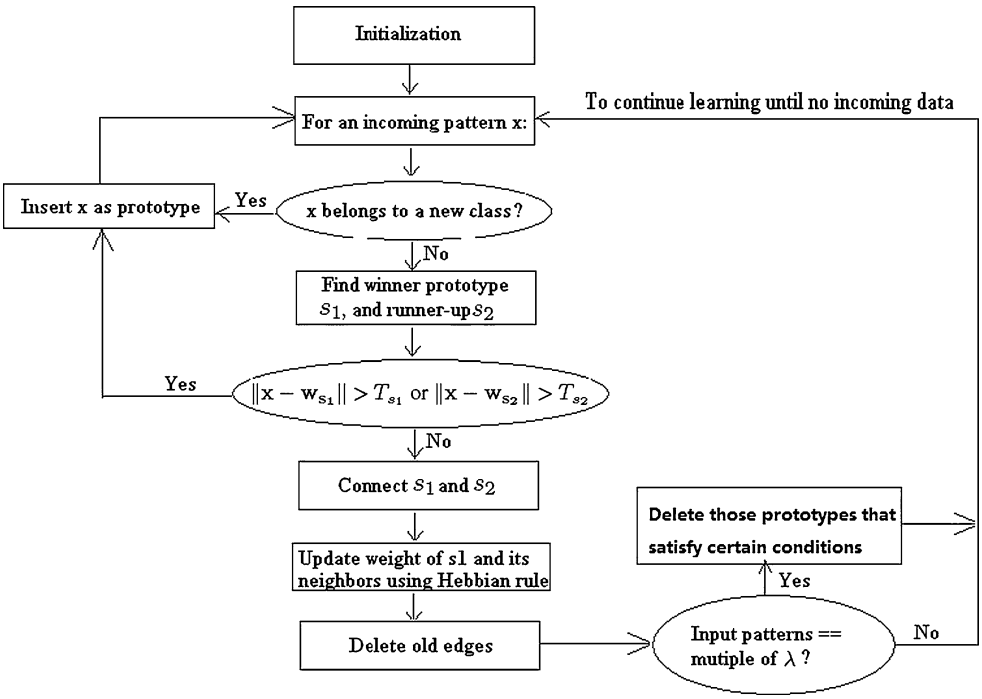

Fig. 1 The overall learning process of ILVQ

At the beginning of the algorithm, we use the first two incoming samples to initialize the prototype set. For every incoming data $\mathbf{x}$, we first justify whether $\mathbf{x}$ belongs to a class that has never been learned before. If $\mathbf{x}$ is from a new class, we accept it as a new prototype to fulfill between-class incremental learning. Otherwise, we find the "winner" and "runner-up" prototypes based on the following equations:

$$winner = argmin_{\mathbf{w_c} \in G} \|\mathbf{x} - \mathbf{w_c}\| \quad (1)$$

$$runnerup = argmin_{\mathbf{w_c} \in G \setminus \{winner\}} \|\mathbf{x} - \mathbf{w_c}\| \quad (2)$$

Therein, G is the set of prototypes.

If a certain insertion condition satisfies, we still insert $\mathbf{x}$ into prototype set to fulfill within-class incremental learning.

Otherwise, if the insertion condition doesn't satisfy, we update the topological relationships and the values of the learned prototypes by input data $\mathbf{x}$. The detailed procedure will be shown later.

For every several learning epoches, we design a condensing scheme to detect the prototypes in low probability density region. Because such prototypes are very likely to be learned from noise patterns, the scheme is helpful to delete those noise prototypes.

In the rest of the section, we analyze ILVQ algorithm in detail. And we will focus on several key issues. (1) Design a policy to decide when and how to incrementally learn and update prototypes. (2) Propose an adaptive threshold technique to automatically learn the proper number of prototypes without any prior knowledge. (3) Apply a condensing scheme to remove the useless prototypes that are very likely to be noises or outliers.

3.1 Incrementally learn and update prototypes

In order to fulfill the incremental learning task, ILVQ ought to grow dynamically. For that reason, we must insert proper prototypes into the prototype set gradually. However, a permanent insertion policy should not be taken: it might result in interminable insertion and overfitting. Instead, we must decide when and how to insert a new prototype and when insertion is to be stopped. To fulfill between-class incremental learning, a pattern from new classes that has not been learned before should be inserted into the prototype set. This policy enables ILVQ to learn prototypes from different classes.

To fulfill within-class incremental learning, we consider the distance between input pattern $\mathbf{x}$ and learned prototypes when determining whether a new prototype ought to be inserted. For patterns from learned classes, we first find the two prototypes "winner" and "runner-up" that are nearest to the input pattern $\mathbf{x}$ based on (1) and (2). If the distance between $\mathbf{x}$ and "winner" prototype is large, it is very likely that pattern $\mathbf{x}$ locates in the border rather than in the center of a class. Border prototypes are more effective than central prototypes in classification tasks [29]. Therefore, in this case we insert $\mathbf{x}$ as a new prototype. Here, we use a threshold T to make the decision: if $|\mathbf{w}_{winner} - \mathbf{x}| > T$, we insert $\mathbf{x}$. To not miss useful border prototypes, we also consider the distance between $\mathbf{x}$ and "runner-up"

prototype. It is because that in this case (the distance between $\mathbf{x}$ and "runner-up" is large), $\mathbf{x}$ is still very likely to locate in the border of a class. To sum up, the insertion operation can be conducted when the distance between $\mathbf{x}$ and $\mathbf{w}_{winner}$ is larger than a certain threshold or when the distance between $\mathbf{x}$ and $\mathbf{w}_{runnerup}$ is larger than the threshold of the runner-up.

$$(\|\mathbf{w}_{winner} - \mathbf{x}\| > T_{winner}) \vee (\|\mathbf{w}_{runnerup} - \mathbf{x}\| > T_{runnerup}) \tag{3}$$

This insertion scheme ensures that ILVQ fulfill between-class and within-class incremental learning.

As discussed in the beginning of this section, pattern $\mathbf{x}$ is used to update the learned prototypes if the insertion condition does not satisfy. In ILVQ, we improve the competitive learning rule proposed by Kohonen [13] to update the value of prototype vectors. The "winner", which is closest to the input pattern $\mathbf{x}$, will be moved based on the input pattern. However, simply moving the winner prototype is insufficient to obtain a sound learning result. Based on the assumption that the prototypes in the neighbor region of winner whose label differs from the winner's are likely to be noise interrupting the prototype set, we update these prototypes as well. First, we divide the prototypes in the neighbor region of the winner into two sets C_1 and C_2 according to their label values (Therein, the neighbor region of a prototype $\mathbf{s}$ means all the prototypes which are connected with $\mathbf{s}$). The prototypes that have the same label as $\mathbf{x}$ are put into C_1, while the remaining prototypes are put into C_2. We update the value of prototypes in this way: when $label_{winner} = label_x$, we update the value of winner and prototypes in set C_2, but don't update the prototypes in C_1. In this case, the direction that these prototypes move will differ from the winner: if the winner is moved towards $\mathbf{x}$, these prototypes will be moved far from $\mathbf{x}$. Therefore, the distance between each prototype in C_2 and the winner will be larger, which is beneficial for us to distinguish patterns from different classes. Otherwise, if $label_{winner} \neq label_x$, we only update the winner and move those prototypes in C_1 with the same direction as the "winner" prototype.

3.2 Learn proper prototype number by an adaptive threshold

As discussed in Sect. 3.1, the distance threshold is important in the algorithm. It determines whether an input pattern should be inserted into the prototype set or not: if the distance between an input pattern $\mathbf{x}$ and the "winner" is larger than the distance threshold of winner T_{winner} or the distance between $\mathbf{x}$ and "runner-up" is larger than $T_{runnerup}$, we will insert $\mathbf{x}$ as a new prototype.

Because the prototype set is growing gradually, the threshold should not be fixed as the accuracy control threshold in Adaptive Resonance Theory [2]; otherwise the prototypes in the prototype set might increase endlessly. In ILVQ, we design a self-adaptive distance threshold technique. The threshold is self-adjustable rather than fixed constantly. We adjust the value of T_i when prototype i becomes the "winner" or "runner-up" of an incoming pattern.

The distance threshold T_i of prototype i ought to be less than the between-class distance of i to distinguish the prototypes from different classes, and larger than the within-class distance of i to avoid ruling out potential useful prototypes in the prototype set. To realize this policy, we set the average distance between i and other prototypes in its neighborhood that have the same label with i as i's within-class distance, and set the minimum distance between i and prototypes in i's neighborhood whose labels differ from i's as the between-class distance.

In Algorithm 1, we specifically state the algorithm used to compute the distance threshold T_i of prototype i.

Algorithm 1 Compute the Distance Threshold T_i

1: Compute the within-class distance $T_{i_{within}}$ by $T_{i_{within}} = \frac{1}{N_{label_i}} \sum_{(i,j)\in E \bigwedge label_i = label_j} \|w_i - w_j\|$.
2: Find the minimum between-class distance $T_{i_{between}} = \min_{(k_1,i)\in E \bigwedge label_i \neq label_{k_1}} \|w_i - w_{k_1}\|$.
3: **if** $T_{i_{between}} < T_{i_{within}}$ **then**
4: Set $T_{i_{between}}$ with the second minimum between-class distance:
5: $T_{i_{between}} = \min_{(k_2,i)\in E \bigwedge label_i \neq label_{k_2} \bigwedge k_1 \neq k_2} \|w_i - w_{k_2}\|$.
6: **end if**
7: Go to step 2 to update $T_{i_{between}}$ until $T_{i_{between}}$ is no less than $T_{i_{within}}$.
8: Set $T_i = T_{i_{between}}$.
9: **return** T_i

3.3 A condensing scheme to denoise

Training sets often contain noise. However, many classification methods ignore to take any efforts to handle those probable noise patterns that are interrupted into training sets. Therefore, the learning performance will be affected to some extent. In the proposed ILVQ, we design a condensing scheme to cope with the probable noise patterns.

From the discussion presented in [24], establishing a reasonable topological structure is beneficial for us to detect noise in the prototype set. We apply the topological representing rule [16] taken by Martinetz to connect two prototypes when they become the "winner" and the

Table 1 Notations used in ILVQ

Notation	Description
G	Prototype vector set, used to store prototypes
N_G	Number of prototypes in G
E	Edge set, used to store edges between prototypes
N_E	Number of edges in E
w_i	Value of prototype i
M_i	Winner count of prototype i
T_i	Distance threshold of prototype i
N_i	Set of neighbors of prototype i
$age_{(i,j)}$	Age of the edge that connects prototypei and j
η_1 and η_2	Learning rate

"runner-up" of an incoming pattern, as introduced in Sect. 3.1. Under an online environment, the prototypes in the prototype set can not remain their neighbor relation constantly. The prototypes that are neighboring at an early stage will not be neighboring at a more advanced stage. Therefore, it is necessary to remove connections that have not been updated recently.

The prototypes that seldom become "winner" or "runner-up" during learning process are likely to be noises or outliers. As a result, it is necessary for us to delete those probable noise. Here, we use the strategy described in [24]: for every several epochs of learning, we try to remove those prototypes that have only one or no topological neighbor. In [24], this strategy works well for removing nodes caused by noise without adding much computational load.

3.4 Summary of ILVQ

Using the analysis described above, we describe the complete algorithm in Algorithm 2. The notations used in the algorithm are shown in Table 1.

Algorithm 2 Incremental Learning Vector Quantization

1: Initialize G to contain the first two input data from the training set.
2: Initialize E storing connection between prototypes to empty set.
3: Input a new pattern $\mathbf{x} \in R^n$.
4: Search set G to find the winner s_1 and runner-up s_2 by $s_1 = argmin_{c\in G}\|\mathbf{x} - w_c\|$ and $s_2 = argmin_{c\in G\setminus\{s_1\}}\|\mathbf{x} - w_c\|$.
5: **if** ($\mathbf{x} \in$ a new class) $\bigvee \|\mathbf{x} - \mathbf{w_{s_1}}\| > T_{s_1} \bigvee \|\mathbf{x} - \mathbf{w_{s_2}}\| > T_{s_2}$ **then**
6: Insert $\mathbf{x}$ into set G.
7: Goto Step3 to process the next input pattern.
8: **end if**
9: **if** $(s_1, s_2) \notin E$ **then**
10: Add edge (s_1, s_2) to edge set E by $E \leftarrow E\bigcup\{(s_1, s_2)\}$.
11: Set the age of edge (s_1, s_2) to zero.
12: **end if**
13: **for** s_i in the neighbor area of s_1 **do**
14: Update $age_{(s_1,s_i)} \leftarrow age_{(s_1,s_i)} + 1$
15: **end for**
16: Update winner count $M_{s_1} \leftarrow M_{s_1} + 1$.
17: **if** $label_{\mathbf{x}} = label_{s_1}$ **then**
18: Update $w_{s_1} \leftarrow w_{s_1} + \eta_1(\mathbf{x} - w_{s_1})$
19: **for** s_i in the neighbor area of node s_1 and $label_{s_i} \neq label_{\mathbf{x}}$ **do**
20: Update $w_{s_i} \leftarrow w_{s_i} - \eta_2(\mathbf{x} - w_{s_i})$
21: **end for**
22: **else**
23: Update $w_{s_1} \leftarrow w_{s_1} - \eta_1(\mathbf{x} - w_{s_1})$
24: **for** s_i in the neighbor area of node s_1 and $label_{s_i} = label_{\mathbf{x}}$ **do**
25: Update $w_{s_i} \leftarrow w_{s_i} + \eta_2(\mathbf{x} - w_{s_i})$
26: **end for**
27: **end if**
28: Delete those edges in set E whose age outstrips the parameter $AgeOld$.
29: **if** the iteration step index is the integer multiple of parameter λ **then**
30: Delete the nodes s_i in set G that have no neighbor node.
31: Delete the nodes s_i whose neighbor node is 1 and $M_{s_i} < 0.5\sum_{j=1}^{N_G}\frac{M_j}{N_G}$.
32: **end if**
33: Goto step 3 to continue the online learning process.
34: **return** set G and set E.

In Algorithm 2, we initialize the prototype set G with first two input patterns from training set. The edge set E is initially empty. For a new pattern $\mathbf{x}$ that comes into the network, if $\mathbf{x}$ comes from a new class, we will add $\mathbf{x}$ into prototype set G at once. Otherwise, we find the winner prototype s_1 and the runner-up s_2. If the respective distances separating $\mathbf{x}$ and s_1 or s_2 are greater than distance threshold T_{s_1} or T_{s_2}, the input pattern $\mathbf{x}$ is regarded as a new prototype. Thus we insert it into set G and process the next pattern that comes from training set.

If the above insertion conditions don't satisfy, $\mathbf{x}$ is not new prototype and we adopt it to adjust the prototype set. If the connection between s_1 and s_2 does not exist, we connect them and add it to edge set E. The initial value of $age_{(s_1,s_2)}$ is set as zero. The age of all edges emanating from s_1 are updated, and the winner count M_{s_1} of s_1 is updated by

$$M_{s_1} = M_{s_1} + 1 \tag{4}$$

To adjust the value of prototypes, the winner s_1 should be moved close to or far away from input pattern $\mathbf{x}$ given to their labels. But simply moving the winner prototype is not sufficient to obtain a sound learning result. We believe that the prototypes in the neighbor region of winner whose label is different from the winner's are very likely to be a noise

interrupted into the prototype set. Therefore, we also update these prototypes according to Kohonen's rule. In this case, the direction that these prototypes move will be exactly different from the winner: if the winner is moved towards **x**, those prototypes will be moved far from **x**. Hence the distance between those prototypes and winner will be larger. It is beneficial for us to distinguish those patterns from different classes. The improved learning rule is presented as follows: if $label_{\mathbf{x}} = label_{s_1}$, then we adjust the winner prototype, and the prototypes s_i that satisfy $(s_1, s_i) \in E$ and $label_{s_1} \neq label_{\mathbf{x}}$. On the contrary, if $label_{\mathbf{x}} \neq label_{s_1}$, then we adjust the winner, and the prototypes s_i that satisfy $(s_1, s_i) \in E$ and $label_{s_1} = label_{\mathbf{x}}$.

To update the neighbor relationship between prototypes during the online learning, we delete the edges in set E whose age is larger than $AgeOld$ (a user-defined parameter). To eliminate the influence of noise, for every several epoch of training, we try to delete the prototypes which have only one or no neighbor prototype. However, for one-dimensional data the prototypes may form a chain, the above scheme will repeatedly remove the boundary. As a result, we add a condition to control the deletion policy: if a prototype s_i has one neighbor node and its winner count $M_{s_i} < 0.5 \sum_{j=1}^{N_G} \frac{M_j}{N_G}$ (In this case, the prototype s_i is very likely to be a noise), we delete it.

The time complexity of the whole algorithm is $O(NDM)$, where N is the number of training data, D is the dimensionality of the training data, and M is the learned prototype number finally. The time complexity of the proposed ILVQ is only linear with the total number of training data, which is competitive compared with some other prototypes algorithms such as LVQ and G-LVQ. As for the storage efficiency, the proposed ILVQ incrementally learns with training data. It means that, instead of storing the whole training set, we only need to load one training data each time, but needn't to store them. Therefore, the proposed ILVQ is very promising to cope with large data sets.

3.4.1 Learning rate

In Algorithm 2, the learning rate η_1 and η_2 will affect the extent to which the winner prototype and the neighbors of the "winner" move nearer or further away from the input pattern.

The Growing Neural Gas (GNG) algorithm [8] applies a constant learning rate. However, it will result in network instability after several epochs of training.

An exponential decaying learning rate is proposed in [11]: $\eta(n) = \eta_0 \exp(-\frac{n}{\tau})$, where η_0 is the initial value of η and τ is the total number of training samples. However, we are still unable to give a standard η_0 for every task. More importantly, we are unable to avail the total amount of training samples τ in an online environment.

In this study, we adopt a method used in [24] to adjust the learning rate over the training epoch by $\eta_1 = \frac{1}{M_{winner}}$ and $\eta_2 = \frac{1}{100 M_{winner}}$. In this case, the learning rate is time variant and we can make the position of the prototype more stable when it becomes a winner more and more times during training. Because of the divergence of the series $\sum_{n=1}^{\infty} \frac{1}{n}$ and the convergence of $\sum_{n=1}^{\infty} \frac{1}{n^2}$, the learning speed of our algorithm satisfies stochastic approximation [27] conditions, i.e. the learning rate decays slowly, but not too slow. The relative slowly decaying learning rate ensures the stability of the algorithm. But the plasticity of the algorithm may be influenced if the learning rate decays too slowly. Thus the stochastic approximation condition insures that ILVQ make a nice compromise between plasticity and stability.

4 Experiment

4.1 Artificial dataset

In this section, we conduct the experiment using a two-dimensional Dataset, as shown in Fig. 2. The sample set has five classes. Class 1 and class 2 satisfy a 2-D Gaussian distribution. Class 3 and class 4 are two concentric rings. Class 5 is a sinusoidal curve. As Fig. 2 shows, 20% noise (patterns obey $2-$ dimensional uniform distribution but are recorded in a label chosen randomly from 1 to 5.) of useful data are distributed on the whole dataset.

4.1.1 Experiment in a stationary environment

First, we apply Fig. 2 to simulate a stationary dataset: 50000 patterns are selected randomly from all classes 1–5.

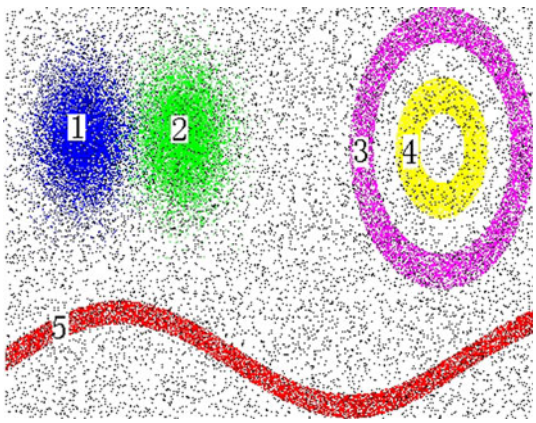

Fig. 2 The original artificial dataset. Data are divided into five classes: class 1 and class 2 are two Gaussian distributions; class 3 and class 4 are two concentric rings; and class 5 is a sinusoidal curve. Furthermore, noise is added to the whole dataset

Here, traditional LVQ and generalized LVQ [23] (an improved version of LVQ) are compared with ILVQ. For ILVQ, we set parameters as $\lambda = AgeOld = 16$ by a 10-fold cross validation policy. The number of prototypes is learned automatically. To clarify the comparison, we adopt three different prototype numbers for LVQ and G-LVQ: $5 \times 8 = 40$, $5 \times 20 = 100$, and $5 \times 40 = 200$. Each class has the same number of prototypes in LVQ and G-LVQ. Each method is repeated three times: for three different sequences in choosing input patterns from training set. We present the average recognition rate in Table 2. Table 2 shows that LVQ might perform more poorly when it has even more prototypes. It might result from bad choice of initial position of prototypes. It demonstrates that the recognition performance of LVQ and G-LVQ will be affected by its initial condition. For ILVQ, the recognition rate of three-time tests is nearly the same; even with fewer prototypes, the recognition performance of ILVQ is much better than either LVQ or G-LVQ. During training, the number and value of prototypes are determined automatically by the algorithm itself. When training is finished, each class has a different number of prototypes: classes 1, 2, 3, 4, and 5 respectively have 8, 9, 11, 6, and 6 prototypes averagely. The prototypes are connected by edges to represent the topological structure of input data, as portrayed in Fig. 3.

From Fig. 3, we know that each class is represented very well by automatically generated prototypes: the noise is eliminated. For edges in Fig. 3, prototypes not only within the same class but also between classes are connected. We highlighted between-class connections as black lines in Fig. 3. Some patterns are positioned between classes; this is the reason for error classification.

4.1.2 Experiment in an incremental environment

Then, we simulate incremental learning using the following scheme: from epoch 1 to 10,000, data are chosen randomly from class 1. At epoch 10,001, we change the data environment and randomly choose samples from class 2.

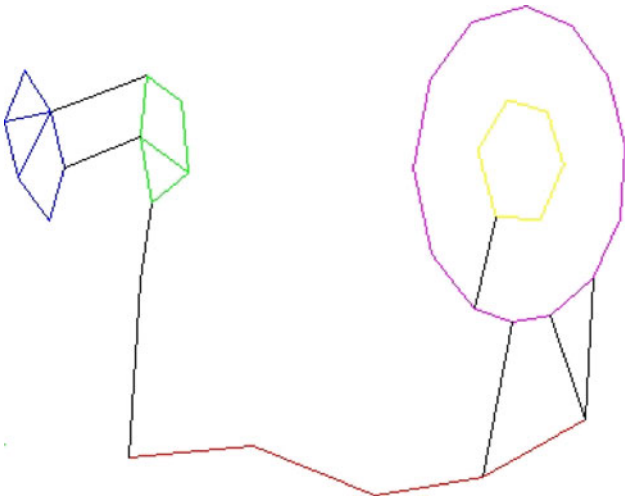

Fig. 3 The learned topological structure of ILVQ: within-class connections in class 1 are expressed as *blue lines*; within-class connections in class 2 are expressed as *green lines*; within-class connections in class 3 are expressed as *magenta lines*; within-class connections in class 4 are expressed as *yellow lines*; and the within-class connections in class 5 are expressed as *red lines*. The between-class connections are highlighted as *black lines*

Similarly, we choose patterns of class 3 from epoch 20,001 to 30,000. At epoch 30,001, we use patterns from class 4. Eventually, from epoch 40,001 to 50,000, we adopt data of class 5 to train the network. We also add 20% noise to the training data.

Actually, ILVQ, G-LVQ and LVQ are repeated three times with different sequences of incoming data to achieve an average result in this incremental environment. The parameters of ILVQ are set as $\lambda = AgeOld = 7$ by a 10-fold cross validation policy. For LVQ and G-LVQ, the amount of prototypes is predefined (40, 100, and 200); each class has the same number. The recognition rates of the three methods are presented in Table 3. For ILVQ, the average prototype numbers for classes are 9, 10, 7, 6, and 8, respectively. From Table 3 we know that LVQ and G-LVQ performs poorly in an incremental data environment because the learned weight vector is often destroyed after

Table 2 Recognition rate(*RR*) for a stationary environment in the artificial data set

Algorithm	No. of prototypes	Recognition rate (%)
ILVQ	40	98.3
LVQ	40	96.3
LVQ	100	95.3
LVQ	200	95.6
G-LVQ	40	97.5
G-LVQ	100	98.1
G-LVQ	200	98.1

Table 3 Recognition rate (*RR*) for the incremental environment in the artificial data set

Algorithm	No. of prototypes	Recognition rate (%)
ILVQ	40	99.3
LVQ	40	44.3
LVQ	100	57.4
LVQ	200	55.8
G-LVQ	40	68.2
G-LVQ	100	74.6
G-LVQ	200	59.2

storing new patterns in the network. However, ILVQ can accommodate incremental tasks well.

4.2 Handwritten digits recognition

4.2.1 Experiment in a stationary environment

Now we use Optical digits from the UCI Machine Learning Repository [1] to test the proposed algorithm. In this dataset, the training set includes 3,823 samples from ten classes; the test set includes 1,797 samples from ten classes. For ILVQ, we apply a tenfold cross-validation policy to tune the parameters. We repeat this policy on the training data three times to obtain the proper value of parameters: $\lambda = AgeOld = 450$. For LVQ and G-LVQ, we predetermine three different quantities of prototypes (200, 300, and 500); the prototype number for each class is the same. We repeat the algorithms eight times with different sequences of incoming data to obtain an average result. The results are presented in Table 4.

The best recorded recognition performance under a stationary data environment in this Optical Digits dataset is 98.0% of the Nearest Neighbors Classifier (NNC) [9]. Actually, LibSVM with a Radius Basis Function as the kernel function achieved a recognition rate of 96.6% and a compression ratio of 31.3%. Passerini et al. [19] improved the SVM using different kernel functions; the best recognition performance he obtained was 97.4%. From Table 4, it is apparent that the recognition performance of ILVQ is near that of the best classifier; it is better than some typical methods such as LVQ and SVM. In addition, the compression ratio of the trained classifier, i.e., the number of prototypes divided by the size of the dataset, is rather sound. Moreover, the number of prototypes for each class generated after learning is different in ILVQ rather than remaining the same in LVQ or G-LVQ. For that reason, ILVQ can use fewer prototypes to represent a simple class, and use more prototypes to represent a complex class.

Table 4 The average recognition rate and compression ratio for stationary environment in optical digits data set

Algorithm	No. of prototypes	Recognition rate (%)	Compression ratio (%)
ILVQ	228	97.5 ± 0.5	6.0 ± 0.1
LVQ	200	96.7 ± 0.5	5.2
LVQ	300	97.0 ± 0.3	7.9
LVQ	500	97.3 ± 0.4	13.1
G-LVQ	200	96.1 ± 0.2	5.2
G-LVQ	300	96.4 ± 0.2	7.9
G-LVQ	500	96.5 ± 0.1	13.1

Patterns from each class might be subject to different distributions. Therefore, it is not proper to represent them with the same number of prototypes. However, it is extremely difficult to predetermine a suitable number of prototypes for each class to represent them with no prior knowledge, as do many other classifiers. Results of this experiment demonstrate that ILVQ can generate a proper number of prototype vectors automatically and completely for each class given to the pattern distribution in training set: averagely class 1 has 27 prototypes; class 2 has 17 prototypes; class 3 has 19 prototypes; class 4 has 29 prototypes; class 5 has 17 prototypes; class 6 has 21 prototypes; class 7 has 22 prototypes; class 8 has 22 prototypes; class 9 has 31 prototypes; and class 10 has 23 prototypes.

Another key point is that the recognition performance of ILVQ is not sensitive to its parameters. We list the recognition rate of ILVQ under different parameters in Table 5. From Table 5, it is apparent that the recognition rate and compression ratio of ILVQ never fluctuate drastically when the parameters change. Moreover, from Table 4 we can know when λ and $AgeOld$ vary from different values, the recognition rate of the proposed ILVQ is better than LVQ and G-LVQ under most conditions.

4.2.2 Experiment in an incremental environment

Here, we conduct another experiment in an incremental environment in order to illustrate another advantage of ILVQ. Similar to Section 3.1.2, we simulate the incremental environment in this way: from step 1 to 5000, we randomly select patterns from class 1; from step 5001 to 10000, we select patterns from class 2; etc. In fact, 10-fold cross-validation is still used to achieve the parameter of ILVQ as $\lambda = AgeOld = 450$. For LVQ and G-LVQ, The total number of prototypes is predetermined and the number for each class is the same. We repeat the experiments eight times with different sequences to obtain an average result. Table 6 presents the results, which demonstrate that in an incremental environment LVQ and G-LVQ can only recognize the patterns from the very class (class 10) that was input into the network last. The learned patterns (from class 1 to class 9) were destroyed when they learned patterns from class 10. However, ILVQ can deal with the incremental data environment well because the learned patterns can always be preserved soundly in the network.

Table 5 Recognition rate and compression ratio of ILVQ under different parameters in optical digits data set: here we always assign $AgeOld$ and λ the same value

($AgeOld$,λ)	200	300	400	450	500	600	700
Recognition rate (%)	97.0	97.2	97.1	97.5	97.4	97.1	97.4
Compression ratio (%)	5.25	6.05	6.07	6.00	6.20	6.57	6.65

Table 6 The average recognition rate and compression ratio for incremental environment in optical digits data set

Algorithm	No. of prototypes	Recognition rate (%)	Compression ratio (%)
ILVQ	291	97.7 ± 0.6	7.6 ± 0.2
LVQ	200	10.0 ± 0.0	5.2
LVQ	300	10.0 ± 0.0	7.9
LVQ	500	10.0 ± 0.0	13.1
G-LVQ	200	10.0 ± 0.0	5.2
G-LVQ	300	10.0 ± 0.0	7.9
G-LVQ	500	10.0 ± 0.0	13.1

4.3 Face recognition

Here, we use the facial images in the dataset of Yale Face to validate the recognition performance of ILVQ (http://cvc.yale.edu/projects/yalefaces/yalefaces.html). Yale Face contains 15 distinct subjects with 11 different images of every subject. For ILVQ, we apply ten-fold cross-validation policy to tune the parameters: $\lambda = AgeOld = 84$. LVQ and GLVQ are also used as a comparison. Three times ten-fold cross-validation is adopted to achieve the average result that is listed in Table 7. It dictates ILVQ can achieve the best classification rate in face recognition.

4.4 Remaining UCI datasets

We use the datasets of Glass, Ionosphere, Iris, Liver disorders, Pima Indians, and Wine in UCI [1] sequentially to test the proposed algorithm. The overall condition of these data sets is shown in Table 8. To evaluate the proposed method better, aside from LVQ and G-LVQ, we compare it to famous prototyped-based classifiers such as the K-Means Classifier KMC(M) [15], the Nearest Subclass Classifier NSC(k) [25], the Multiscale Data Condensation algorithm MDC(k) [17], and the Nearest Neighbor Classifier NNC(k). All the above classifiers except NNC are unable to cope with an incremental learning task. For that reason, we do the experiment in a stationary environment. First, we also adopt a three-times 10-fold cross-validation policy to tune the parameters in each algorithm. Because of the small sample number of these datasets, we apply a 10-fold cross-validation procedure to estimate the recognition performance. We also repeat tenfold cross-validation three times, i.e., for three independent draws of 10 non-overlapping subsets from the training set and report the average recognition rate. The patterns are selected randomly from the training set each time. The recognition rates are shown in Table 9; some experimental results are obtained from [25]. The parameters achieved in each method and compression ratio are presented in Table 10. The best and near best performances are emphasized with figures in bold typeface.

Table 7 Recognition rate in Yale Face

Algorithm	ILVQ	LVQ	LVQ	LVQ	G-LVQ	G-LVQ	G-LVQ
No. of prototypes	64	66	88	110	66	88	110
Recognition rate (%)	90.0	84.5	85.5	86.4	85.5	85.5	86.4

Table 8 Overview of the data sets used in the experiments

Dataset	No. of samples	No. of features	No. of classes
Glass	214	9	6
Ionosphere	351	34	2
Iris	150	4	3
Liver disorders	345	6	2
Pima Indians	768	8	2
Wine	178	13	3

From Table 9 we know that, for Glass, Iris, Liver, and Pima datasets, the recognition performance of ILVQ is best or nearly the best. For all databases, the average recognition rate of ILVQ is higher than those of all other classifiers. For the Ionosphere dataset, ILVQ works slightly worse than NSC but better than other classifiers. Table 10 shows that the compression ratio of NSC for Ionosphere is 31%, which is much larger than ILVQ (25.6%). To the aspect of compression ratio, ILVQ is better than NSC. For the Wine dataset, although ILVQ performs slightly worse than NSC, MDC, and NNC, all three classifiers have a 100% or nearly 100% compression ratio; the compression ratio of ILVQ is 12%. Additionally, not only LVQ, but also LVQ-family classifiers are unable to obtain a remarkable recognition performance in this dataset. For example, traditional LVQ obtains 72.3% ± 1.5% and G-LVQ can achieve 72.9% ± 4.8% in wine.

From the perspective of the compression ratio, ILVQ is still better than LVQ and G-LVQ; it is also better than NSC and MDC, which are specially designed to reduce data storage. Although the average compression ratio of ILVQ (15.3%) is slightly higher than KMC of 11.7%, the recognition performance of ILVQ (79.1%) is much better than KMC (73.7%). Through the analysis described above ILVQ can be recognized as presenting the best compromise between recognition performance and compression ratio.

Moreover, we present the results of ILVQ in the incremental data environment in Table 11. It is apparent that the performance of ILVQ in an incremental environment is as good as that in a stationary environment. We do not compare ILVQ with other methods under an incremental

Table 9 Recognition rate (%) of experiments

Dataset	ILVQ	LVQ	G-LVQ	KMC	NSC	MDC	NNC
Glass	**73.3 ± 0.5**	68.3 ± 2.0	**72.8 ± 0.8**	68.8 ± 1.1	70.2 ± 1.5	**73.1 ± 0.7**	**72.3 ± 1.2**
Ionosphere	89.5 ± 0.2	86.4 ± 0.8	88.6 ± 0.9	87.4 ± 0.6	**91.9 ± 0.8**	86.0 ± 0.7	86.1 ± 0.7
Iris	**97.1 ± 0.9**	96.1 ± 0.6	**96.7 ± 0.3**	96.2 ± 0.8	96.3 ± 0.4	95.3 ± 0.4	**96.7 ± 0.6**
Liver	**67.3 ± 1.3**	66.3 ± 1.9	**67.4 ± 1.5**	59.3 ± 2.3	62.9 ± 2.3	61.0 ± 1.5	**67.3 ± 1.6**
Pima	**73.7 ± 1.0**	73.5 ± 0.9	71.1 ± 1.0	58.7 ± 0.9	68.6 ± 1.6	67.9 ± 1.7	**74.7 ± 0.7**
Wine	73.5 ± 4.1	72.3 ± 1.5	72.9 ± 4.8	71.9 ± 1.9	**75.3 ± 1.7**	**75.2 ± 1.7**	73.9 ± 1.9
Average	**79.1 ± 1.3**	77.2 ± 1.3	78.3 ± 1.6	73.7 ± 1.3	77.5 ± 1.4	76.7 ± 1.1	78.5 ± 1.1

Table 10 Parameters and compression ratios (%) of experiments

Dataset	ILVQ			LVQ		G-LVQ		KMC		NSC		MDC		NNC	
	CR	λ	*AgeOld*	*CR*	M	*CR*	M	*CR*	M	*CR*	σ^2_{max}	*CR*	k	*CR*	k
Glass	**25.5**	786	140	45	97	48.7	105	**17**	6	97	0.005	100	1	100	1
Ionosphere	25.6	525	525	**6.8**	24	34	120	**4.0**	7	31	1.25	100	1	100	2
Iris	19.9	21	17	15	22	22.5	33	**8.0**	4	**7.3**	0.25	**9.3**	5	100	14
Liver	**6.7**	16	18	8.4	29	20.9	72	11	19	**4.9**	600	100	1	100	14
Pima	**2.3**	90	13	3.4	26	2.6	20	**1.0**	4	**1.7**	2600	8.1	4	100	17
Wine	12	199	94	32	57	10.1	18	29	17	96	4.0	100	1	100	1
Average	**15.3**			18.4		23.1		**11.7**		39.7		69.6		100	

Table 11 Results of ILVQ in an incremental environment

Dataset	Recognition rate	Compression ratio	λ	*AgeOld*
Glass	71.4 ± 1.9	29.8	786	140
Ionosphere	89.1 ± 0.2	31.7	530	243
Iris	97.3 ± 0.7	21.9	66	66
Liver	66.6 ± 2.5	4.5	5	63
Pima	73.2 ± 3.3	1.8	6	55
Wine	73.9 ± 2.5	11.3	124	85
Average	78.6 ± 1.9	16.8		

environment because other classifiers, aside from NNC, are unable to realize incremental learning.

From results of the experiments above, we conclude that ILVQ performs perfectly in all the three performance categories of recognition rate, compression ratio, and incremental learning.

5 Conclusion

As described in this paper, we proposed an online incremental prototype classifier. The proposed method, designated as incremental learning vector quantization (ILVQ), grows gradually and stores the learned prototypes perfectly so that it can realize the incremental learning task well. It takes advantage of a threshold-based insertion criterion to automatically generate the number of prototypes for each class. Consequently, users no longer need to predetermine it. An innovative method is proposed to adjust the value of the threshold dynamically. Moreover, a proper condensing scheme is taken to ensure ILVQ to eliminate the noise that comes into the network during learning.

In the experiments, we compared ILVQ with some typical prototype classifiers. The results show that the recognition rate of ILVQ will be better under most conditions. In the respect of compression ratio, the performance of ILVQ is also good. Although some algorithms such as KMC achieves a better compression ratio, it must be emphasized that ILVQ fulfills the best compromise between classification performance and storage efficiency. Moreover, the experiments demonstrate that ILVQ can perform the incremental learning task well.

Acknowledgments This work was supported in part by the Fund of the National Natural Science Foundation of China (Grant No. 60975047, 60723003, 60721002), 973 Program (2010CB327903), and Jiangsu NSF grant (#BK2009080).

References

1. Blake CL, Merz CJ (1996) UCI repository of machine learning databases. University of California Department of Information, Irvine

2. Carpenter G, Grossberg S (1990) Hierarchical search using chemical transmitters in self-organizing pattern recognition architectures. Neural Netw 3(3):129–152
3. Cauwenberghs G, Poggio T (2002) Incremental and decremental support vector machine learning. In: NIPS02
4. Dasarathy BV (1991) Nearest neighbor NN norms: NN Pattern Classification Techniques. IEEE Computer Society Press
5. Devijver P, Kittler J (1982) Pattern recognition: a statistical approach. Prentince-Hall, NJ
6. Duda R, Hart D, Stork P (2001) Pattern classification, 2nd edn. Wiley Press, NY
7. Eick CF, Zeidat N, Vilalta R (2004) Using representative-based clustering for nearest neighbor dataset editing. In: ICDM04, pp 375–378
8. Fritzke B (1995) A growing neural gas network learns topologies. In: NIPS95, pp 625–632
9. Gates G (2004) The reduced nearest neighbor rule. IEEE Trans Inf Theory 18(3):431–433
10. Hastie T, Tibshirani R, Friedman J (2001) The elements of statistical learning: data mining, inference, and prediction. Springer
11. Haykin S (2004) A comprehensive foundation, 2nd edn. China Machine Press, China
12. Kim SW, Oommen BJ (2005) On using prototype reduction schemes and classifier fusion strategies to optimize kernel-based nonlinear subspace methods. IEEE Trans PAMI 27(3):455–460
13. Kohonen T (1990) Improved versions of learning vector quantization. In IJCNN90, pp 545–550
14. Lia B, Xuea X, Fan J (2001) Learn++: an incremental learning algorithm for supervised neural networks. IEEE Trans SMC Part C 31:497–508
15. Likas S, Vlassis N, Verbeek J (2005) The global k-means clustering algorithm. Pattern Recogn 36(2):451–461
16. Martinetz TM, Schulten K (1994) Topology representing networks. Neural Netw 7(3):507–522
17. Mitra P, Murthy CA, Pal SK (2002) Density-based multiscale data condensation. IEEE Trans PAMI 24(6):734–747
18. Mollineda RA, Ferri FJ, Vidal E (2002) A merge-based condensing strategy for multiple prototype classifiers. IEEE Trans SMC, Part B 32(5):662–668
19. Passerini A, Frasconi P (2002) From margins to probabilities in multiclass learning problems. In: ECAI02
20. Pekalska E, Duin RPW (2008) Beyond traditional kernels: classification in two dissimilarity-based representation spaces. IEEE Trans SMC, Part C 38(6):729–744
21. Polikar R, Honavar V (2001) Learn++: an incremental learning algorithm for supervised neural networks. IEEE Trans SMC Part C 31:497–508
22. Ren C-X, Dai D-Q (2010) Incremental learning of bidirectional principal components for face recognition. Pattern Recogn Lett 43:318–330
23. Sato A, Yamada K (1995) Generalized learning vector quantization. In: NIPS95, pp 424–429
24. Shen F, Hasegawa O (2008) A fast nearest neighbor classifier based on self-organizing incremental neural network. Neural Networks 21(10):1537–1547
25. Veenman CJ, Reinders MJT (2005) The nearest subclass classifier: a compromise between the nearest mean and nearest neighbor classifier. IEEE Trans PAMI 27(9):1417–1429
26. Villegas M, Paredes R (2008) Simultaneous learning of a discriminative projection and prototypes for nearest-neighbor classification. In: CVPR08, pp 1–8
27. Wasan MT (1969) Stochastic approximation. Cambridge University Press, Cambridge
28. Wilson DL (1972) Asymptotic properties of nearest neighbor rules using edited data. IEEE Trans SMC, Part B 2(5):408–420
29. Wu M, Scholkopf B (2007) A local learning approach for clustering. In: NIPS07, pp 1529–1536
30. Zhang H, Berg AC, Maire M, Malik J (2006) Svm-knn: Discriminative nearest neighbor classification for visual category recognition. In: CVPR06, pp 2126–2134
31. Zhang H, Malik J (2003) Learning a discriminative classifier using shape context distances. In: CVPR03, pp 242–247
32. Zhang J, Gruenwald L (2006) Opening the black box of feature extraction: Incorporating visualization into high-dimensional data mining processes. In: ICDM06, pp 18–22