# An Enhanced Incremental Prototype Classifier using Subspace Representation Scheme

Ye Xu[1], Furao Shen[1], Jinxi Zhao[1], and Osamu Hasegawa[2]

[1] National Key Laboratory for Novel Software Technology, Nanjing University
yexu@smail.nju.edu.cn; {frshen; jxzhao}@nju.edu.cn
[2] Imaging Science and Engineering Lab, Tokyo Institute of Technology
hasegawa@isl.titech.ac.jp

**Abstract.** Prototype classifiers have been studied for many years. But most methods adopt single vectors as prototypes to represent the original data. It is difficult to learn the local information [1] with such prototypes. In this paper, we propose an incremental classifier named Subspace Based Prototype Classifier (SBPC) to handle the issue. SBPC designs an augmented strategy to enhance traditional prototype classifiers. Instead of using single vector, each prototype of SBPC represents a subset of input data using a subspace. And we employ an incremental subspace representation method to learn a subspace for each prototype. By designing a self-adaptive threshold policy, SBPC automatically learns the number and value of prototypes without any prior knowledge. Through adopting both condensing scheme and editing scheme [3], the prototypes are incremental learned, automatically adjusted (condensing scheme) and removed (editing scheme). Results of experiments described herein show that the proposed SBPC accommodates the non-stationary data environment and provides good recognition performance and storage efficiency.

**Key words:** Prototype Classifier, Subspace Based Prototype, Incremental Learning, Subspace Representation

## 1 Introduction

Prototype based method [7] 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. [4] 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 [29], visualization recognition system [30][28], optimization [10], and subspace learning [21].

Nearest Neighbor Classifier (NNC) [2] 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 of learning prototypes have been proposed recently.

One class of prototype methods is classifier based on condensing scheme [3]. 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) [6]. It is based on k-means prototypes positioning with 1-NN rule. Learning Vector Quantization (LVQ) [11], along with some LVQ family methods such as Generalized Learning Vector Quantization (GLVQ) [15] and Incremental Learning Vector Quantization (ILVQ) [26], takes advantage of class information to move the prototype vectors, in order to improve the quality of the classifier decision region. Mollineda et. al. [13] adopt geometric property and clusters to search a condensed set of prototypes. Pekalska and Duin [14] 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 [24] removes all examples that have been misclassified by 1-NN rule from training sets. Eick et. al. [5] propose a editing scheme using supervised clusters. Nearest Subclass Classifier (NSC) [20] applies a technique to merge prototypes based on a defined variance constraint parameter.

But all these typical methods have some shortcomings. (1) Almost all of the reported prototype classifiers merely use single vectors to represent prototypes; local learning [1] has not been considered. Therefore, for these methods, local information is not able to be completely represented by vector based prototypes; the generalization performance may be affected to some extent [19]. (2) Many methods need users to predetermine the number of prototypes based on some prior knowledge. However, such knowledge is not available sometimes; an improper prototype number will affect the classification performance. (3) Most classifiers adopt either condensing scheme or editing scheme. Few methods combine the two schemes. As a result, storage efficiency and noise detection issues can not be addressed at the same time. (4) Many of those algorithms are batch methods. Such methods can not handle non-stationary data environment, rendering it difficult in real time application tasks.

In this paper, we try to introduce an incremental classification algorithm named as Subspace Based Prototype Classifier (SBPC) to handle the discussed shortcomings of typical prototype classifiers. The primary contribution of the proposed method is the augmented strategy introduced to enhance traditional "vector" based prototype classifiers. Instead of using a single vector, we apply a subspace to describe the local geometry of input data. Here, we employ an incremental subspace representation method – Incremental Orthogonal Basis Anal-

ysis (IOBA) [27] to represent the local geometry. For each prototype, the base vectors of the subspace are incrementally learned and automatically adjusted. Therefore, the set of prototypes can effectively represent both global and local characteristics of original data space, which benefits the classification capacity. The second contribution of SBPC is that, based on a self-adaptive threshold, the number of prototypes is automatically determined by the distribution of original data space without any prior knowledge. Users need not to predetermine the number and the initial value of prototypes. The third contribution of the proposed method is the combination of condensing scheme and editing scheme. In our work, the prototypes are incremental learned, automatically adjusted (condensing scheme) and removed (editing scheme) according to the distribution of the training data. Last but not least, SBPC realizes online learning. Thus it can be used in non-stationary environment effectively.

We organize the rest of this paper as follows. In section 2, SBPC is specifically introduced. In section 3, we will do experiments to compare SBPC with some typical prototype classifiers to show the efficiency of our approach.

## 2 Subspace Based Prototype Classifier

As mentioned in section 1, we try to propose a Subspace Based Prototype Classifier (SBPC) in order to fulfill the following goals. (1) To enhance traditional prototype classifier by substituting subspace based prototypes for single vector based prototypes. (2) To automatically learn the number of prototypes needed without any prior knowledge and to adaptively adjust it according to the distribution of training data. (3) To combine condensing scheme and editing scheme during learning process. (4) To realize online learning.

To fulfill the above targets, we give the derivation of algorithm and then highlight the key aspects of the construction and update of the subspace based prototypes, the self-adaptive threshold policy, and the condensing and editing schemes.

### 2.1 Algorithm Derivation

Let us consider a dataset $\{(\mathbf{x_i}, y_i)\}_{i=1}^{N}$. Therein, $\mathbf{x_i} \in \mathbb{R}^{d\times 1}$ is the input pattern, $y_i$ is the label of input pattern, and $N$ is the size of training set. The goal of the proposed method is to select a set of subspace based prototypes $\{(\mathbf{B_j}, l_j)\}_{j=1}^{m}$ that minimize the following error function:

$$E = \sum_{j=1}^{N} (g(\mathbf{x_i}) - y_i)^2 \tag{1}$$

Therein, $g(\mathbf{x_i})$ is the computed label of $\mathbf{x_i}$ obtained by:

$$g(\mathbf{x_i}) = \underset{l_j}{\operatorname{argmin}}\, ProDist(\mathbf{x_i}, \mathbf{B_j}) \tag{2}$$

Therein, $l_j$ is the label of prototype $\mathbf{B_j}$, and $ProDist(\mathbf{x_i}, \mathbf{B_j})$ means the projection distance of pattern $\mathbf{x_i}$ onto the subspace of prototype $\mathbf{B_j}$:

$$ProDist(\mathbf{x_i}, \mathbf{B_j}) = \|\mathbf{x_i} - \sum_{k=1}^{l} (\mathbf{b_k^{(j)}})^\top \mathbf{x_i} \mathbf{b_k^{(j)}}\|_2 \tag{3}$$

where $\{\mathbf{b_k^{(j)}}\}_{k=1}^{l}$ are basis of the subspace of prototype $\mathbf{B_j}$.

For each new incoming pattern $\mathbf{x_i}$, we find the "winner" prototype $\mathbf{B_{winner}}$ by

$$\mathbf{B_{winner}} = \underset{\mathbf{B_j}}{\operatorname{argmin}}\, ProDist(\mathbf{x_i}, \mathbf{B_j}) \tag{4}$$

and the "runner-up" prototype $\mathbf{B_{runnerup}}$ by

$$\mathbf{B_{runnerup}} = \underset{\mathbf{B_j} \wedge \mathbf{j} \neq \mathbf{winner}}{\operatorname{argmin}}\, ProDist(\mathbf{x_i}, \mathbf{B_j}). \tag{5}$$

Then, we set up topological relationship among prototypes: "winner" prototype $\mathbf{B_{winner}}$ and "runner-up" $\mathbf{B_{runnerup}}$ are connected if a connection between them does not exist. The topological structure among prototypes is helpful for us to detect noise. (This point will be discussed later.)

If the label of $\mathbf{x_i}$ is different from it of the "winner", i.e., the current prototype set misclassifies $\mathbf{x_i}$ according to the supposition that training data and test data are subject to the same distribution, we will use $\mathbf{x_i}$ to create a new prototype, in order to minimize the value of $(g(\mathbf{x_i}) - y_i)^2$. Also, the distance between prototypes should be taken into account when we determine whether a new prototype ought to be added. If the projection distance of $\mathbf{x_i}$ onto "winner" prototype $\mathbf{B_{winner}}$ is large, it is very likely that pattern $\mathbf{x_i}$ locates in the border rather than in the center of a class. Here, we use a threshold $T$ to make the decision: if $ProDist(\mathbf{x_i}, \mathbf{B_{winner}}) > T$, we insert a new prototype. Border prototypes are more effective than central prototypes in classification tasks [24]. To not miss useful border prototypes, we also consider the projection distance of $\mathbf{x_i}$ onto $\mathbf{B_{runner-up}}$. It is because that in this case (the projection distance of $\mathbf{x_i}$ onto $\mathbf{B_{runner-up}}$ is large), $\mathbf{x_i}$ is still very likely to locate in the border of a class. To sum up, we use $\mathbf{x_i}$ to construct a new prototype if:

$$y_i \neq l_{winner} \vee ProDist(\mathbf{x_i}, \mathbf{B_{winner}}) > T \vee ProDist(\mathbf{x_i}, \mathbf{B_{runnerup}}) > T \tag{6}$$

Otherwise, we update learned "winner" and those prototypes in the neighbor of "winner" using pattern $\mathbf{x_i}$ in order to minimize $ProDist(\mathbf{x_i}, \mathbf{B_j})$ if these prototypes have the same label as $\mathbf{x_i}$'s. In this way, the value of cost function $E$ is reduced.

For every several epoch of learning, we adopt an editing scheme to delete those useless prototypes and connections between prototypes.

In the rest of section 2, we focus on several key issues of SBPC: (1) Design a scheme to learn subspace for each prototype. (2) Propose a threshold technique to learn the proper number of prototypes according to the original data distribution. (3) Develop a method to incrementally learn useful prototypes (condensing scheme) and dynamically remove those prototypes (editing scheme) that are likely to be noise or outliers.

### 2.2 Learn a Subspace for Each Prototype

Most reported prototype algorithms represent original data with sets of single vectors. Such methods only focuses on global learning [19], but fails to learn local information. Hence, the generalization performance is affected especially when the distribution of patterns in the input space is uneven [25]. In order to cope with this issue, we propose an augmented strategy: learning a subspace for each prototype. Therefore, for SBPC, the set of prototypes reflect the overall condition of data distribution, and each prototype can represent the local distribution of input data.

**Subspace Representation Method: Incremental Orthogonal Basis Analysis** We need a subspace representation algorithm to learn subspace for each prototype. Principal Component Analysis (PCA) [3] is the most popular subspace representation method. It searches for several component vectors from original data along which variations are extremal. Recently, many subspace representation methods have been proposed based on different purposes. For example, Candid Convariance-free Incremental Principal Component analysis (CCIPCA) [23] can be used for online learning.

In this paper, we employ a new incremental subspace representation algorithm named Incremental Orthogonal Basis Analysis (IOBA) [27]. It aims at learning a group of components (base vectors) for feature space from original data. IOBA has several characteristics that are very proper for SBPC to incrementally learn a subspace for each prototype: (1) IOBA learns numerically orthogonal presentation (a group of orthogonal base vectors) according to a series of input patterns. Orthogonal base vectors contain little redundant information; it renders SBPC efficient in learning the local data distribution. (2) In IOBA, the target dimension of subspace is just data-dependent. It means we need not to predetermine the number of base vectors for the subspace using any prior knowledge. (3) IOBA is an incremental method. As patterns comes one by one in online manner, the base vectors of subspace are incrementally learned and automatically adjusted. Thereby, the prototypes in SBPC can be incrementally learned and dynamically adjusted during learning.

Here, we give the detailed IOBA in Algorithm 1.

In Algorithm 1, we first initialize the basis $\mathbf{B}$ to the empty basis, and the initial dimension of $\mathbf{B}$ is 0. For each incoming pattern $\mathbf{x_j}$, we apply the online Gram–Schmidt orthogonalization method to generate a potential basis vector $\mathbf{b_{k+1}}$. But unlike traditional Gram–Schmidt method, we consider the independence between input data $\mathbf{x_j}$ and currently learned components $\mathbf{b_1}, \mathbf{b_2}, ..., \mathbf{b_k}$ before decide whether to accept the computed potential basis vectors. It is because that if a base vector is transformed from a pattern that has strong dependence on learned basis, the computation error would arise and affect the orthogonality among base vectors [9]. Thus if we accept $\mathbf{b_{k+1}}$ as a new base vector for feature space without considering the independence between input data $\mathbf{x_j}$ and learned components, the numerical orthogonality between $\{\mathbf{b_1}, \mathbf{b_2}, ..., \mathbf{b_k}\}$ and $\mathbf{b_{k+1}}$ can't be guaranteed.

**Input**: $d$ dimension input patterns $\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_N}, ...$

1: Initialize the basis $\mathbf{B} = \emptyset$ and the basis dimension $k = dim(\mathbf{B}) = 0$.
2: For each input new pattern $\mathbf{x_j}(j > k)$:
3: **for** $i = 1$ to $k$ **do**
4: Compute $r_{i,k+1}$ by $r_{i,k+1} = \mathbf{x_j}^\top \mathbf{b_i}$.
5: **end for**
6: Compute $\hat{\mathbf{b}}_{\mathbf{k+1}}$ by $\hat{\mathbf{b}}_{\mathbf{k+1}} = \mathbf{x_j} - \sum_{i=1}^{k} r_{i,k+1}\mathbf{b_i}$.
7: Compute $r_{k+1,k+1}$ by $r_{k+1,k+1} = \|\hat{\mathbf{b}}_{\mathbf{k+1}}\|_2$.
8: Compute potential base vector $\mathbf{b_{k+1}}$ by $\mathbf{b_{k+1}} = \frac{\hat{\mathbf{b}}_{\mathbf{k+1}}}{r_{k+1,k+1}}$.
9: **if** $\frac{r_{k+1,k+1}}{\|\xi_{k+1}\|} \geq \frac{dim(\mathbf{B})}{d}$ (A self-adaptive threshold technique that guarantees the numerical orthogonality of learned base vectors) **then**
10: Accept $\mathbf{b_{k+1}}$ as a base vector and add it into basis $\mathbf{B}$.
11: Update basis dimension $k \leftarrow k + 1$.
12: **end if**
13: Goto step 3 to continue the learning process.

**Output**: Basis dimension $dim(\mathbf{B})$ and all base vectors $\mathbf{b_1}, \mathbf{b_2}, ..., \mathbf{b_k}$.

**Algorithm 1**: Incremental Orthogonal Basis Analysis

According to this, we employ an independence measure and then select potential components to ensure the numerical orthogonality of learned components (basis).

Based on linear dependence theorem [9], we measure the numerical independence between input data $\mathbf{x_{k+1}}$ and currently learned components $\mathbf{b_1}, \mathbf{b_2}, ..., \mathbf{b_k}$ by the projection distance of $\mathbf{x_{k+1}}$ onto subspace $span\{\mathbf{b_1}, \mathbf{b_2}, ..., \mathbf{b_k}\}$:

$$\|\xi_{\mathbf{k+1}} - span\{\mathbf{b_1}, \mathbf{b_2}, ..., \mathbf{b_k}\}\|_2 \tag{7}$$

The distance in (7) can be rewritten in the following form:

$$\min_{a_1, a_2, ..., a_k} \|\xi_{\mathbf{k+1}} - (a_1\mathbf{b_1} + a_2\mathbf{b_2} + ... + a_k\mathbf{b_k})\|_2 \tag{8}$$

where $\alpha = (a_1, a_2, ..., a_k)^\top$ is a coordinate vector.

We define matrix $\mathbf{B_k} = [\mathbf{b_1}, \mathbf{b_2}, ..., \mathbf{b_k}] \in \mathbb{R}^{d \times k}$, thus (7) can be rewritten in the form:

$$\min_{\alpha} \|\mathbf{B_k}\alpha - \xi_{\mathbf{k+1}}\|_2 \tag{9}$$

The column vectors in $\mathbf{B_k}$ are orthogonal. Thereby, based on $\mathbf{B_k}$, we can easily construct an orthogonal matrix $\mathbf{Q_2}$ in the following form:

$$\mathbf{Q_2} = [\mathbf{B_k}, \mathbf{Q_1}] \tag{10}$$

Therein, $\mathbf{Q_2} \in \mathbb{R}^{d \times d}$, $\mathbf{Q_1} \in \mathbb{R}^{d \times (d-k)}$, and the column vectors of $\mathbf{Q_1}$ and $\mathbf{B_k}$ are orthogonal.

Since orthogonalization transformation never changes the 2-norm of a vector, (9) is equal to the following expression:

$$\|\mathbf{B_k}\alpha - \xi_{\mathbf{k+1}}\|_2 = \|\mathbf{Q_2}^\top(\mathbf{B_k}\alpha - \xi_{\mathbf{k+1}})\|_2 = \| \begin{pmatrix} \alpha \\ \mathbf{0} \end{pmatrix} - \begin{pmatrix} \mathbf{B_k}^\top \xi_{\mathbf{k+1}} \\ \mathbf{Q_1}^\top \xi_{\mathbf{k+1}} \end{pmatrix} \|_2 \tag{11}$$

We let $\alpha = \mathbf{B_k}^\top \xi_{\mathbf{k+1}}$. Thereby, the value of expression (7) is equal to $\|\mathbf{Q_1}^\top \xi_{\mathbf{k+1}}\|_2$, i.e., the independence between learned feature space and input vector $\xi_{k+1}$ can be measured by the value of $\|\mathbf{Q_1}^\top \xi_{\mathbf{k+1}}\|_2$. The theorem below will denote the relationship between the 2-norm of vector $\mathbf{Q_1}^\top \xi_{\mathbf{k+1}}$ and the value of $r_{k+1,k+1}$.

**Theorem 1** *The 2-norm of* $\mathbf{Q_1}^\top \xi_{\mathbf{k+1}}$ *shown in (11) is equal to the value of* $r_{k+1,k+1}$ *achieved in the* $(k+1)^{th}$ *iteration of IOBA, i.e.,* $r_{k+1,k+1} = \|\mathbf{Q_1}^\top \xi_{\mathbf{k+1}}\|_2$.

The proof is quite straightforward and we omit it here for brevity.

So we can measure the independence between the learned basis $\{\mathbf{b_i}\}_{i=1}^{k}$ and the input vector $\xi_{\mathbf{k+1}}$ via the value of $r_{k+1,k+1}$. If $r_{k+1,k+1}$ is large, i.e., the independence between $\mathbf{b_{k+1}}$ and learned feature space $span\{\mathbf{b_1}, \mathbf{b_2}, ..., \mathbf{b_k}\}$ is strong, we accept $\mathbf{b_{k+1}}$ into basis $M$ of feature space. Otherwise, we reject computed vector $\mathbf{b_{k+1}}$. Here, we adopt a threshold policy: if $r_{k+1,k+1} \geq T$, we accept potential component $\mathbf{b_{k+1}}$ into basis $M$. And we set the value of threshold $T = \frac{dim(\mathbf{B})}{d}$, so the threshold is self-adaptive.

According to this threshold policy, we only accept those potential base vector that are transformed from patterns strongly independent of learned basis. Thus the numerical orthogonality of learned basis can be guaranteed. Moreover, the difficulty of accepting components increases with the increase of the basis dimension $dim(\mathbf{B})$ for feature space. It is very helpful to achieve a nice storage efficiency for subspace learning methods.

At last, we should point out that the value of $\frac{dim(\mathbf{B})}{d}$ is confined to the interval of $[0, 1]$. Therefore, we use $\frac{r_{k+1,k+1}}{\max_i r_{ii}}$ instead of $r_{k+1,k+1}$ for comparison convenience given that the value of $\frac{r_{k+1,k+1}}{\max_i r_{ii}}$ is definitely between 0 and 1.

**Learn Subspace by IOBA** As mentioned in section 2.1, we need to insert a new prototype if (6) satisfies. Here, we adopt $\frac{\mathbf{x_i}}{\|\mathbf{x_i}\|_2}$ as the base vector of the subspace for the new constructed prototype. During the online learning process, we use IOBA to adjust the subspace of constructed prototypes in prototype set.

For each prototype in prototype set, if a prototype $\mathbf{B_j}$ becomes the "winner" prototype of an incoming pattern $\mathbf{x_k}$, and condition (6) does not satisfy, we use $\mathbf{x_k}$ to update subspace of $\mathbf{B_j}$ by IOBA. According to the process of IOBA, we know that if $\mathbf{x_k}$ is orthogonal of current subspace of $\mathbf{B_j}$, we accept a new base vector that is directly computed by $\mathbf{x_k}$. On the contrary, if the threshold condition does not satisfy, which means $\mathbf{x_k}$ is dependent on subspace of $\mathbf{B_j}$, we need to do nothing. According to this analysis, we know that after this update process, $\mathbf{x_k}$ is absolutely contained in the subspace of $\mathbf{B_j}$, i.e.,

$$\mathbf{x_k} \in span\{\mathbf{b_1^{(j)}}, \mathbf{b_2^{(j)}}, ..., \mathbf{b_l^{(j)}}\} = \{\sum_{i=1}^{l} t_i \mathbf{b_i^{(j)}}; t_i \in \mathbb{R}^1\} \tag{12}$$

Therein, $\{\mathbf{b_i^{(j)}}\}_{i=1}^{l}$ are base vectors of the subspace of prototype $\mathbf{B_j}$.

Condition (6) does not satisfy, thus prototype $\mathbf{B_j}$ and input pattern $\mathbf{x_k}$ share the same label. Therefore, we can conclude that after the update process, $ProDist(\mathbf{x_k}, \mathbf{B_j})$ is reduced, guaranteeing that $(g(\mathbf{x_k}) - y_k)^2$ remains zero.

For those prototypes $\mathbf{B_s}$ which locate in the neighborhood of winner and share the same label with $\mathbf{x_i}$, we still use $\mathbf{x_i}$ to update subspace of $\mathbf{B_s}$ by IOBA. Because under this circumstance, pattern $\mathbf{x_i}$ is in the local area of $\mathbf{B_s}$, and it is necessary for such prototypes to learn some local information.

In Algorithm 2, we give the detailed procedure of learning subspace based prototypes. Due to the subspace based prototype policy, the proposed method learns not only global data distributions by the whole set of prototypes, but also the local knowledge by each single prototype. It means that SBPC achieves a nice compromise between globality and locality.

**Input**: Input data with label $(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), ..., (\mathbf{x_N}, y_N), ...$

For each input new pattern $(\mathbf{x_k}, y_k)$:
**if** (6) satisfies **then**
  Construct a new prototype and adopt $\frac{\mathbf{x_i}}{\|\mathbf{x_i}\|_2}$ as the base vector of the subspace for the constructed prototype.
**else**
  **if** ($\mathbf{B_j}$ is winner of $\mathbf{x_k}$) **then**
    Use pattern $\mathbf{x_k}$ to update the subspace of $\mathbf{B_j}$ via IOBA.
  **end if**
  **if** ($\mathbf{B_s}$ locates in the neighbor of winner) $\wedge$ $(y_k = l_s)$ **then**
    Use pattern $\mathbf{x_k}$ to update the subspace of $\mathbf{B_s}$ via IOBA.
  **end if**
**end if**
Goto step 1 to continue the learning process.

**Output**: Dimension $dim(\mathbf{B_j})$ and basis $\{\mathbf{b_i^{(j)}}\}_{i=1}^{k}$ for the subspace of each prototype $\mathbf{B_j}$.

**Algorithm 2**: Learning Subspace for Each Prototype

### 2.3 Learn Proper Prototype Number Using Self-adaptive Threshold

As mentioned in section 2.1, we insert a new prototype if (6) satisfies. Therefore, the threshold $T$ in (6) is very important to determine the number of prototypes.

The network is gradually growing and self-adjusted; it is not wise to set up a constant threshold for each prototype. To properly insert prototypes for SBPC, the value of threshold for each prototype should not remain the same. Therefore, we can conclude that $T_j$ should be a self-adaptive threshold that can be automatically adjusted according to the data. In SBPC, we adjust threshold $T_j$ of prototype $\mathbf{B_j}$ when $\mathbf{B_j}$ becomes the "winner" or "runner-up".

The threshold $T_j$ of prototype $\mathbf{B_j}$ ought to be larger than the within-class distance of $\mathbf{B_j}$ to avoid ruling out useful prototypes in the network. Also, $T_j$ needs

to be less than the between-class distance of $\mathbf{B_j}$ to distinguish the prototypes from different classes. Here, within class distance means the average value of distance between $\mathbf{B_{winner}}$ and those prototypes that share the same label with $\mathbf{B_{winner}}$; between-class distance of prototype $\mathbf{B_j}$ means the distance between $\mathbf{B_j}$ and those prototypes in the neighborhood of $\mathbf{B_j}$ that have different label from $\mathbf{B_j}$'s.

In Algorithm 3 we specifically state the algorithm to compute the distance threshold $T_j$ of prototype $\mathbf{B_j}$.

1: Compute the within-class distance $DWithin_j$ by
$DWithin_j = \frac{1}{N_{label_i}} \sum_{(i,j)\in E \bigwedge l_i = l_j} Dist(\mathbf{B_i}, \mathbf{B_j})$.
2: Find the minimum between-class distance
$DBetween_j = \min_{(k_1,j)\in E \bigwedge l_j \neq l_{k_1}} Dist(\mathbf{B_j}, \mathbf{B_{k_1}})$.
3: **if** $DBetween_j < DWithin_j$ **then**
4: Set $DBetween_j$ with the second minimum between-class distance:
5: $DBetween_j = \min_{(k_2,j)\in E \bigwedge l_j \neq l_{k_2} \bigwedge k_1 \neq k_2} Dist(\mathbf{B_j}, \mathbf{B_{k_2}})$.
6: **end if**
7: Go to step2 to update $DBetween_j$ until $DBetween_j \geq DWithin_j$.
8: Set $T_j = DBetween_j$.
9: **return** $T_j$

**Algorithm 3**: Compute the distance threshold $T_j$ of prototype $\mathbf{B_j}$

We need to find a proper measurement to measure the distance between two subspaces of corresponding prototypes in Algorithm 3. And we hope that the distance measurement should have several characteristics. (i) The used measurement must be a metric, namely, it satisfies non-negativity, symmetry, and triangle inequality. (ii) Because we adopt the metric to compute the distance between two subspaces, we hope that the distance is relatively small if the two subspaces are similar, and the distance is large if two subspaces are different. (iii) The used distance between two subspaces should be compatible with the projection distance defined by (3).

According to the above analysis, we adopt the $L_2 - Hausdorff$ [22] distance to measure the distance between two prototypes $\mathbf{B_{w_1}}$ and $\mathbf{B_{w_2}}$:

$$Dist(\mathbf{B_{w_1}}, \mathbf{B_{w_2}}) = max(\boldsymbol{d}(\mathbf{B_{w_1}}, \mathbf{B_{w_2}}), \boldsymbol{d}(\mathbf{B_{w_2}}, \mathbf{B_{w_1}})) \tag{13}$$

Therein, $\boldsymbol{d}(\mathbf{B_{w_1}}, \mathbf{B_{w_2}})$ is the directional distance from $m-$dimensional subspace $\mathbf{B_{w_1}}$ to $n-$dimensional subspace $\mathbf{B_{w_2}}$:

$$\boldsymbol{d}(\mathbf{B_{w_1}}, \mathbf{B_{w_2}}) = \sqrt{m - \sum_{i=1}^{m} \sum_{j=1}^{n} \|\mathbf{b_i^{(w_1)}} - \mathbf{b_j^{(w_2)}}\|_2^2} \tag{14}$$

Therein, $\{\mathbf{b_i^{(w_1)}}\}_{\mathbf{i=1}}^{\mathbf{m}}$ are the basis of the subspace of prototype $\mathbf{B_{w_1}}$, and $\{\mathbf{b_j^{(w_2)}}\}_{\mathbf{j=1}}^{\mathbf{n}}$ are the basis of the subspace of prototype $\mathbf{B_{w_2}}$.

In Theorem 2, we specify that the $L_2 - Hausdorff$ distance satisfies characteristics i), ii) and iii).

**Theorem 2** *The $L_2 - Hausdorff$ distance shown in (13) satisfies (i), (ii), and (iii).*

*Proof.* Firstly, it is apparently that $L_2 - Hausdorff$ distance satisfies non-negativity and symmetry. The triangle inequality is demonstrated in [17]. Therefore, $L_2 - Hausdorff$ is a metric.

Secondly, if two subspaces $\mathbf{B_{w_1}}$ and $\mathbf{B_{w_2}}$ overlap completely, the dimension of the two subspaces are equal, and base vectors of each subspace $\{\mathbf{b_i^{(w_1)}}\}_{i=1}^{m}$ and $\{\mathbf{b_j^{(w_2)}}\}_{j=1}^{n}$ are totally the same. Under this circumstance, the $L_2 - Hausdorff$ distance between $\mathbf{B_{w_1}}$ and $\mathbf{B_{w_2}}$ are 0. On the contrary, if subspace $\mathbf{B_{w_1}}$ is perpendicular to $\mathbf{B_{w_2}}$, i.e., the base vectors for each subspace are mutually orthogonal, $Dist(\mathbf{B_{w_1}}, \mathbf{B_{w_2}})$ reaches the maximum value: $max(m, n)$.

Thirdly, note that the projection distance of a normalized pattern $\xi$ onto subspace $\mathbf{B}$:

$$ProDist(\xi, \mathbf{B}) = \|\xi - \sum_{i=1}^{n} \mathbf{b_i}^\top \xi \mathbf{b_i}\|_2 = \sqrt{(\xi - \sum_{i=1}^{n} \mathbf{b_i}^\top \xi \mathbf{b_i})^\top (\xi - \sum_{i=1}^{n} \mathbf{b_i}^\top \xi \mathbf{b_i})} \quad (15)$$

$$= \sqrt{\xi^\top \xi - \sum_{i=1}^{n} (\mathbf{b_i}^\top \xi)^2} = \sqrt{1 - \sum_{i=1}^{n} (\mathbf{b_i}^\top \xi)^2} = \boldsymbol{d}(span\{\xi\}, \mathbf{B}) \quad (16)$$

Equation (15) and (16) mean that the projection of pattern $\xi$ onto subspace $\mathbf{B}$ is equal to the directional distance from $1-$dimensional subspace $span\{\xi\}$ to the $n-$dimensional subspace $\mathbf{B}$. Therefore, the $L_2 - Hausdorff$ distance is compatible with the projection distance defined by (2).

Due to the three characteristics, $L_2 - Hausdorff$ distance is able to measure the distance between two subspaces well.

### 2.4 Incremental Method Using both Condensing and Editing Scheme

To fulfill the incremental task, the set of prototypes should grow incrementally. Therefore, we should gradually insert prototypes into the set of prototypes. However, permanent insertion policy should not be taken: it might result in waste of storage capacity and overfitting. Instead, we must apply certain criterion to decide when and how to inset a new prototype and when the insertion ought to be stopped. As discussed in section 2.1, we only insert a new prototype when the input pattern is misclassified by the current prototype set or the pattern is very likely to locate in the border of a class:

$$y_i \neq l_{winner} \vee ProDist(\mathbf{x_i}, \mathbf{B_{winner}}) > T_{winner} \vee ProDist(\mathbf{x_i}, \mathbf{B_{runnerup}}) > T_{runnerup} \quad (17)$$

Therein, $\mathbf{B_{winner}}$ and $\mathbf{B_{runnerup}}$ are "winner" and "runner-up" prototypes that are found by (4) and (5) respectively. The condensing technique can reduce the error function $E$ shown in (1) according to the supposition that training data and test data are subject to the same distribution.

Training patterns may contain noise. During the learning process, we may use such noise data to learn prototypes. Thereby, we apply an editing scheme to handle this issue. Because establishing a topological relationship among prototypes is beneficial to detect noise prototypes [16], we build a topological structure among learned prototypes. Enlightened by Martinetz topological representing rule [12], we connect two prototypes when they become the "winner" and "runner-up" prototype of an incoming pattern. To cater to incremental learning, the topological structure should be dynamically updated. The prototypes that are neighboring at an early stage may not be neighboring at a more advanced stage. Therefore, it is necessary to remove connections that have not been updated for a long time. Based on the dynamical topological structure, we delete prototypes that may be noise: for every several epochs of learning, we remove those prototypes that have only one or no topological neighbor. The prototype that seldom becomes "winner" or "runner-up" is very likely to be a noise, so that this strategy can work effectively for removing prototypes caused by noise.

### 2.5 Summary of SBPC

Here, we give the detailed Subspace Based Prototype Classifier in Algorithm 4.

In the beginning of the proposed method, we initialize prototype set $G$ to contain two $1-$dimensional prototypes. The base vector of subspace for each prototype is initialized with the first two incoming data respectively. We initialize the connection set $E$ that stores connection between prototypes to empty set.

For every incoming data $(\mathbf{x_i}, y_i)$, we find the "winner" and "runner-up" prototypes by (4) and (5). Then we update the threshold of "winner" and "runner-up" using Algorithm 3.

If the "winner"'s label is different from $y_i$, which means the current learned prototype sets misclassified pattern $\mathbf{x_i}$, we use $\mathbf{x_i}$ to construct a new prototype. To make the condensing scheme better, we insert new prototypes if the projection distance of $\mathbf{x_i}$ onto subspace of $\mathbf{B_{winner}}$ is greater than $T_{winner}$ or the projection distance of $\mathbf{x_i}$ onto $\mathbf{B_{runnerup}}$ is greater than $T_{runnerup}$. Because under this condition, $\mathbf{x_i}$ is very likely to locate in the border of a class.

If the above insertion condition doesn't satisfy, we adopt pattern $\mathbf{x_i}$ to update the learned prototypes. First, we set up a connection relationship between "winner" and "runner-up". Then, we use $\mathbf{x_i}$ to update the subspace of "winner" and those prototypes in the neighbor area of winner, as indicated in Algorithm 2. We only update the subspace of those prototypes that share the same label with $\mathbf{x_i}$. Last, we update the winner count of prototype and "age" of connections. Here, winner count records the times that a prototype becomes "winner".

The proposed SBPC is an incremental method, so that the neighborhood relationship among prototypes should not remain constantly. Connections that

**Input**: Input patterns $\{(\mathbf{x_i}, y_i)\}_{i=1}^{N}$.

Initialize prototype set $G$ to contain two prototypes, and use the first two input data to initialize basis of subspace for the two prototypes respectively.
Initialize $E$ storing connection between prototypes to empty set.
For each incoming pattern $(\mathbf{x_i}, y_i)$:
Search set $G$ to find winner $\mathbf{B_{winner}}$ and runner-up $\mathbf{B_{runnerup}}$ by (4) and (5).
Update winner and runner-up's thresholds $T_{winner}$ and $T_{runnerup}$ by Algorithm 3.
**if** (Expression (17) satisfies) **then**
    Create a new prototype and use normalized $\mathbf{x_i}$ as the first base vector of its subspace.
    Goto Step3 to process the next input pattern.
**end if**
**if** $(\mathbf{B_{winner}}, \mathbf{B_{runnerup}}) \notin E$ **then**
    Add edge $(\mathbf{B_{winner}}, \mathbf{B_{runnerup}})$ to edge set $E$ by $E \leftarrow E \bigcup \{(\mathbf{B_{winner}}, \mathbf{B_{runnerup}})\}$.
    Set the age of edge $(\mathbf{B_{winner}}, \mathbf{B_{runnerup}})$ to zero.
**end if**
**for** $\mathbf{B_j}$ in the neighbor area of $\mathbf{B_{winner}}$ **do**
    Update $Age_{(\mathbf{B_{winner}}, \mathbf{B_j})} \leftarrow Age_{(\mathbf{B_{winner}}, \mathbf{B_j})} + 1$
**end for**
Update winner count of $\mathbf{B_{winner}}$ by $WTime_{winner} \leftarrow WTime_{winner} + 1$.
Use $(\mathbf{x_i}, y_i)$ to update subspace of $\mathbf{B_{winner}}$ by IOBA.
**for** Prototype $\mathbf{B_j}$ in the neighbor area of $\mathbf{B_{winner}}$ and $l_j = y_i$ **do**
    Use $(\mathbf{x_i}, y_i)$ to update subspace of $\mathbf{B_j}$ by IOBA.
**end for**
Delete those edges in set $E$ whose age outstrips the parameter *AgeOld*.
**if** The iteration step index is the integer multiple of parameter $\lambda$ **then**
    Delete the prototypes $\mathbf{B_j}$ in set G that have no neighbor prototype.
    Delete $\mathbf{B_j}$ that has 1 neighbor prototype and $WTime_j < 0.5 \sum_{i=1}^{|G|} \frac{WTime_i}{|G|}$.
**end if**
Goto step 3 to continue the online learning process.

**Output**: Prototype set $G$.

**Algorithm 4**: Subspace Based Prototype Classifier

have not been refreshed recently need to be removed. Thus we delete those connections whose age outstrips *AgeOld* (a user defined parameter).

For every $\lambda$ (another user defined parameter) iteration steps, we adopt an editing scheme to remove those noise interrupted into prototype set. It is well known that those prototypes that seldom become "winner" or "runner-up" are very likely to be "noise". Therefore, we eliminate those prototypes that have no neighbor, or have only one neighbor and relatively small winner count.

## 3 Experiment

In this section, we use seven typical datasets: Glass, Ionosphere, Iris, Liver Disorder, Optical Digits, Sonar, and Wine from the UCI Machine Learning Repository [18] to test the proposed algorithm. In Table 1, we give the overview of these datasets.

**Table 1.** Overview of the datasets used in the experiments

| Dataset | No. of samples | No. of features | No. of classes |
|---|---|---|---|
| Optical Digits | 5620 | 64 | 10 |
| Glass | 214 | 9 | 6 |
| Ionosphere | 351 | 34 | 2 |
| Iris | 150 | 4 | 3 |
| Liver disorders | 345 | 6 | 2 |
| Sonar | 208 | 60 | 2 |
| Wine | 178 | 13 | 3 |

**Table 2.** Overall characteristics for each method: "Y" means "Yes"; "N" means "No"; "S" means "Subspace"; "V" means "Vector"

| Algorithm | SBPC | NNC | KMC | LVQ | MDC | NSC | GLVQ | ILVQ |
|---|---|---|---|---|---|---|---|---|
| Prototype | S | V | V | V | V | V | V | V |
| Online learning | Y | N | N | N | N | N | Y | Y |
| Determine prototype no. automatically | Y | Y | N | N | Y | Y | N | Y |
| Use Condensing scheme | Y | N | Y | Y | Y | N | Y | Y |
| Use Editing Scheme | Y | N | N | N | N | Y | N | N |

To evaluate the proposed method better, aside from Nearest Neighbor Classifier (NNC), we compare SBPC with some typical prototype classifiers: Learning Vector Quantization (LVQ), K-Means Classifier (KMC), Multiscale Data Condensation algorithm (MDC), Nearest Subclass Classifier (NSC), Generalized Learning Vector Quantization (GLVQ), and Incremental Learning Vector Quantization (ILVQ). LVQ, KMC, and MDC are typical condensing classifiers; NSC is an editing classifier that is proposed recently. MDC and NSC are specially designed to reduce compression ratio, i.e., the number of prototypes divided by the size of the dataset; GLVQ and ILVQ are two incremental classifiers. We list the overall characteristics of each classifier in Table 2.

### 3.1 Handwritten Digits Recognition

We use Optical Digits from the UCI Machine Learning Repository to test the proposed method. In this database, the training set includes 3823 samples; the

test set includes 1797 samples. During training process, 50000 samples are randomly picked up from training set to learn prototypes. For SBPC, we tune the parameters $\lambda = 3000$ and $AgeOld = 3000$ by ten times 10-fold cross validation policy. After the set of prototypes is learned, we test the label of patterns from the test set to achieve the recognition rate. We repeat the experiment ten times to achieve an average recognition result, as listed in Table 3. Some typical classifiers listed in Table 2 are used to compare with SBPC. The recognition rate and compression ratio are used as the benchmark to compare different classifiers. From Table 3, we know that, under the dataset of Optical Digits, the recognition performance of SBPC is 97.5%. It is better than KMC, LVQ, MDC, NSC, and GLVQ. The recognition rate of SBPC is a little worse than it of NNC (98.0%), which is the best-recorded performance under Optical Digits. But we should note that to the aspect of compression ratio, NNC is very poor (100%), while SBPC is the best (1.8%) among the eight classifiers.

**Table 3.** Recognition rate ($RR$), number of prototypes, and Compression ratio ($CR$) under dataset of Optical Digits

| Algorithm | SBPC | NNC | KMC | LVQ | MDC | NSC | GLVQ | ILVQ |
|---|---|---|---|---|---|---|---|---|
| RR | 97.5% | 98.0% | 90.0% | 97.3% | 96.3% | 97.3% | 96.5% | 97.5% |
| Prototype No. | 70 | 3823 | 100 | 230 | 375 | 271 | 256 | 268 |
| CR | 1.8% | 100% | 2.6% | 6.0% | 9.8% | 7.1% | 6.7% | 7.0% |

**Table 4.** Automatically learned prototype number for each class of SBPC

| Class | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| No. of Prototypes | 4 | 8 | 6 | 8 | 7 | 8 | 5 | 5 | 8 | 11 | 70 |

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. For SBPC, not only the total number of prototypes, but also the number of prototypes for each class are automatically learned according to original data distribution. Experiments show that SBPC generates different number of prototypes for each class, as illustrated in Table 4. It means that SBPC is able to adopt small number of prototypes to represent data of simple classes and use relatively more prototypes to represent complex classes.

### 3.2 Remaining UCI databases

Here, we adopt some other databases from UCI Machine Learning Repository to test the proposed SBPC. Similar to section 3.1, some typical prototype classifiers

are used to make comparison. For each method, we still use ten times ten-fold cross validation policy to tune the parameters. Because these datasets contain small number of samples, we use ten-fold cross validation to test the recognition performance for each classifier. The recognition results are shown in Table 5; the Compression Ratio ($CR$) and the tuned parameters for each method are listed in Table 6. The best and near the best performances are emphasized with figures in bold typeface.

**Table 5.** Recognition Rate of experiments

| Dataset | SBPC | NNC | KMC | LVQ | MDC | NSC | GLVQ | ILVQ |
|---|---|---|---|---|---|---|---|---|
| Glass | 71.4±1.2 | **72.3±1.2** | 68.8±1.1 | 68.3±2.0 | **73.1±0.7** | 70.2±1.5 | **72.8±0.8** | **73.3±0.5** |
| Ionosphere | **93.7±0.7** | 86.1±0.7 | 87.4±0.6 | 86.4±0.8 | 86.0±0.7 | **91.9±0.8** | 88.6±0.9 | 89.5±0.2 |
| Iris | **97.3±1.0** | **96.7±0.6** | **96.2±0.8** | 96.1±0.6 | 95.3±0.4 | 96.3±0.4 | **96.7±0.3** | **97.1±0.9** |
| Liver | **67.1±1.5** | **67.3±1.6** | 59.3±2.3 | 66.3±1.9 | 61.0±1.5 | 62.9±2.3 | **67.4±1.5** | **67.3±1.3** |
| Sonar | **82.0±1.3** | **81.8±1.4** | **81.9±2.5** | 78.3±2.4 | **82.7±1.0** | 81.3±1.1 | 81.5±1.6 | 80.0±1.3 |
| Wine | **77.6±2.2** | 73.9±1.9 | 71.9±1.9 | 72.3±1.5 | **75.2±1.7** | **75.3±1.7** | 72.9±4.8 | 73.5±4.1 |
| Average | **81.5±1.3** | 79.7±1.2 | 77.6±1.5 | 78.0±1.5 | 78.9±1.2 | 79.7±1.3 | 80.0±1.7 | 80.1±1.4 |

**Table 6.** Compression Ratios ($CR$) and tuned parameters of experiments

| Dataset | SBPC ($CR$,$\lambda$,AgeOld) | NNC ($CR$,k) | KMC ($CR$,M) | LVQ ($CR$,M) | MDC ($CR$,k) | NSC ($CR$,$\sigma^2$) | GLVQ ($CR$,M) | ILVQ ($CR$,$M_1$,$M_2$) |
|---|---|---|---|---|---|---|---|---|
| Glass | (42.0,137,137) | (100,1) | (**17**,6) | (45,97) | (100,1) | (97,0.005) | (48.7,105) | (25.5,786,140) |
| Ionosphere | (**1.6**,36,36) | (100,2) | (**4.0**,7) | (6.8,24) | (100,1) | (31,1.25) | (34,120) | (25.6,525,525) |
| Iris | (**4.4**,1,1) | (100,14) | (8.0,4) | (15,22) | (9.3,5) | (7.3,0.25) | (22.5,33) | (19.9,21,17) |
| Liver | (11.1,72,54) | (100,14) | (11,19) | (8.4,29) | (100,1) | (**4.9**,600) | (20.9,72) | (**6.7**,16,18) |
| Sonar | (30.5,216,33) | (100,1) | (**17**,18) | (**19**,40) | (100,1) | (70,0.05) | (74.5,70) | (**14.5**,42,42) |
| Wine | (**13.2**,50,10) | (100,1) | (29,17) | (32,57) | (100,1) | (96,4.0) | (**10.1**,18) | (**12**,199,94) |
| Average $CR$ | **17.1** | 100 | **14.3** | 21.0 | 84.9 | 51.0 | 35.1 | **17.4** |

From Table 5, we can find that, under the datasets of Ionosphere, Iris, Liver, Sonar, and Wine, the proposed SBPC achieves the best or near the best recognition performance. For all the six databases, the average recognition rate of SBPC (81.5%) is the best among the eight classifiers. SBPC only performs a little worse (71.4%) than four other methods under the database of Glass. But Table 6 indicates that most methods that outstrip SBPC under Glass have larger compression ratio than it of SBPC. For NNC and MDC, the two classifiers have a 100% compression ratio; for GLVQ, the compression ratio (48.7) is still larger than it of SBPC.

To the respective to compression ratio, SBPC is averagely better than NNC, LVQ, GLVQ, and ILVQ; it is also better than MDC and NSC, which are spe-

cially designed to reduce compression ratio. Although the compression ratio of SBPC (17.1) is a slightly larger than it of KMC (14.3), it deserves noting that the performance of KMC (77.6%) is far worse than it of SBPC. It is because the number of prototypes learned by KMC is too less to represent the original data space. Therefore, we can conclude that, compared with KMC and other classifiers, SBPC achieves the best compromise between recognition performance and storage efficiency.

## 4 conclusion

As described in this paper, we propose an incremental subspace based prototype algorithm (SBPC) for online classification tasks. By designing an augmented strategy of constructing a subspace to represent local geometry of input data, it enhances the traditional "vector" based prototype. By addressing a self-adaptive threshold policy, SBPC learns not only the total prototype number, but also the number of prototypes for each class according to data distribution; it needs not to use any prior knowledge to learn the number and value of prototypes. By adopting both condensing scheme and editing scheme, SBPC automatically learns new prototypes and adjusts learned prototype set, so that it fulfills incremental learning tasks well.

In the experiment section, we conduct several experiments to test the validity of the proposed method. To compare with SBPC, we use several typical prototype classifiers such as NNC, KMC, LVQ, MDC, NSC, GLVQ, and ILVQ. KMC, LVQ, and MDC are typical condensing classifiers; NSC is an editing classifiers. MDC and NSC are specially proposed for reducing compression ratio; GLVQ and ILVQ are typical incremental methods. The results of experiment show that SBPC obtains the best recognition rate among the eight classifiers. To the aspect of compression ratio, SBPC is better than most compared classifiers. Although KMC works a little better than SBPC in compression efficiency, we can conclude that SBPC achieves the best compromise between classification capacity, storage efficiency, and incremental learning.

## 5 Acknowledgements

The authors would like to thank Wei Chen for some code work. The work was supported in part by the China NSF grant (#60573157, #60723003, and #60775046).

## References

1. Bottou, L., Vapnik, V.: Local Learning Algorithm. Neural Computation. 4(6), 888–900 (1992)
2. Dasarathy, B.V.: Nearest Neighbor NN Norms: NN Pattern Classification Techniques. IEEE Computer Society Press. (1991)

3. Devijver, P., Kittler, J.: Pattern Recognition: A Statistical Approach. NJ: Prentince-Hall. (1982)
4. Duda, R., Hart, P., Stork, D.: Pattern Classification, 2nd Edition. Wiley Press. (2001)
5. Eick, C. F., Zeidat, N., Vilalta, R.: Using Representative-Based Clustering for Nearest Neighbor Dataset Editing. In: ICDM04, pp. 375–378. IEEE Computer Society Press, (2004)
6. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer. (2001)
7. Haykin, S.: Neural networks: a Comprehensive Foundation, 2nd Edition. China Machine Press. (2004)
8. He, X., Cai, D., Han, J.: Learning a Maximum Margin Subspace for Image Retrieval. IEEE Trans. on Knowledge and Data Engineering. 20(2), 189–201 (2008)
9. He, X.: Numerical Dependence Theorem and its Application (in Chinese). Numerical Mathematics, A Journal of Chinese Universities. 1(1), 11–19 (1979)
10. Kim, S. W., Oommen, B. J.: On Using Prototype Reduction Schemes and Classifier Fusion Strategies to Optimize Kernel-Based Nonlinear Subspace Methods. IEEE Trans. on PAMI. 27(3), 455–460 (2005)
11. Kohonen, T.: Improved Versions of Learning Vector Quantization. In: IJCNN90, pp. 545–550. IEEE Computer Society Press, (1990)
12. Martinetz, T. M., Schulten, K.: Topology Representing Networks. Neural Networks. 7(3), 507–522 (1994)
13. Mollineda, R. A., Ferri, F. J., Vidal, E.: A Merge-Based Condensing Strategy for Multiple Prototype Classifiers. IEEE Trans. on Systems, Man and CyberneticsPart B: Cybernetics. 32(5), 662–668 (2002)
14. Pekalska, E., Duin, R. P. W.: Beyond Traditional Kernels: Classification in Two Dissimilarity-Based Representation Spaces. IEEE Trans. on Systems, Man and CyberneticsPart C: Applications and Reviews. 38(6), 729–744 (2008)
15. Sato, A., Yamada, K.: Generalized Learning Vector Quantization. In: NIPS95, pp. 424–429. MIT Press, (1995)
16. Shen, F., Hasegawa, O.: A Fast Nearest Neighbor Classifier Based on Self-organizing Incremental Neural Network. Neural networks. 21(10), 1537–1547 (2008)
17. Sun, X., Wang, L., Feng, J.: Further Results on the Subspace Distance. Pattern Recognition. 40(1), 328–329 (2007)
18. Blake, C. L., Merz, C. J.: UCI Repository of Machine Learning Databases. Irvine, CA: University of California Department of Information (1996)
19. Vapnik, V.: The Nature of Statistical Learning Theory. Springer Verlag, New York. (1995)
20. Veenman, C. J., Reinders M. J. T.: The Nearest Subclass Classifier: A Compromise between the Nearest Mean and Nearest Neighbor Classifier. IEEE Trans. on PAMI. 27(9), 1417–1429 (2005)
21. Villegas, M., Paredes, R.: Simultaneous Learning of a Discriminative Projection and Prototypes for Nearest-Neighbor Classification. In: CVPR08, pp. 1–8. IEEE Computer Society Press, (2008)
22. Wang, L., Wang, X., Feng, J.: Intrapersonal Subspace Analysis with Application to Adaptive Bayesian Face Recognition. Pattern Recognition. 38(4), 617–621 (2005)
23. Weng, J., Zhang, Y., Hwang, W. S.: Candid Covariance-Free Incremental Principal Component Analysis. IEEE Trans. on PAMI. 25(8), 1034–1040 (2003)
24. Wilson, D. L.: Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. IEEE Trans. on Systems, Man, and Cybernetics. 2, 408–420 (1972)

25. Wu, M., Scholkopf, B.: A Local Learning Approach for Clustering. In: NIPS07, pp. 1529–1536. MIT Press, (2007)
26. Xu, Y., Shen, F., Hasegawa, O., Zhao, J.: An Online Incremental Learning Vector Quantization. In: PAKDD09, pp. 1046–1053. Springer, (2009)
27. Xu, Y., Shen, F., Zhao, J., Hasegawa, O.: To Obtain Orthogonal Feature Extraction Using Training Data Selection. In: CIKM09. ACM Press, (2009)
28. Zhang, H., Berg, A. C., Maire, M., Malik, J.: SVM-KNN: Discriminative Nearest Neighbor Classification for Visual Category Recognition. In: CVPR06, pp. 2126–2134. IEEE Computer Society Press, (2006)
29. Zhang, H., Malik, J.: Learning a Discriminative Classifier Using Shape Context Distances. In: CVPR03, pp. 242–247. IEEE Computer Society Press, (2003)
30. Zhang, J., Gruenwald, L.: Opening the Black Box of Feature Extraction: Incorporating Visualization into High-Dimensional Data Mining Processes. In: ICDM06, pp. 18–22. IEEE Computer Society Press, (2006)