Neural Comput & Applic (2013) 22:1023–1035
DOI 10.1007/s00521-011-0793-1

ORIGINAL ARTICLE

An online incremental learning support vector machine for large-scale data

Jun Zheng · Furao Shen · Hongjun Fan · Jinxi Zhao

Received: 23 June 2011 / Accepted: 22 December 2011 / Published online: 5 January 2012

Abstract Support Vector Machines (SVMs) have gained outstanding generalization in many fields. However, standard SVM and most of modified SVMs are in essence batch learning, which make them unable to handle incremental learning or online learning well. Also, such SVMs are not able to handle large-scale data effectively because they are costly in terms of memory and computing consumption. In some situations, plenty of Support Vectors (SVs) are produced, which generally means a long testing time. In this paper, we propose an online incremental learning SVM for large data sets. The proposed method mainly consists of two components: the learning prototypes (LPs) and the learning Support Vectors (LSVs). LPs learn the prototypes and continuously adjust prototypes to the data concept. LSVs are to get a new SVM by combining learned prototypes with trained SVs. The proposed method has been compared with other popular SVM algorithms and experimental results demonstrate that the proposed algorithm is effective for incremental learning problems and large-scale problems.

J. Zheng · F. Shen (✉) · H. Fan · J. Zhao
National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
e-mail: frshen@nju.edu.cn

J. Zheng
e-mail: justinnju@gmail.com

H. Fan
e-mail: fhjcq@nju.edu.cn

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

F. Shen
Jiangyin Information Technology Research Institute, Nanjing University, Nanjing, China

Keywords Online incremental SVM · Incremental learning · Large-scale data

1 Introduction

SVM [1] has established itself as the most widely used kernel learning algorithm, and it has been successfully used for many problems. The good generalization property of an SVM depends on a subset called Support Vectors (SVs), which are sufficient description of the decision bound. Unfortunately, the training of traditional SVM is very time and space consuming for the reason that SVM training usually is a quadratic programming (QP) problem. Thus, it may be intractable for traditional SVM to deal with large-scale problems.

According to [2], there are three obvious ways in which we can solve large-scale problem. The first way is to treat the data as a stream, then apply online algorithms. In this paper, we call these algorithms online incremental algorithm, which means in batch learning environment, part of data is presented once. The second one is to use parallel algorithm. The last one is to sample a small subset from the original data, i.e., scale down the problem size.

To apply SVMs for large-scale problems, one way is to adapt SVMs to learn incrementally. Adapting learning methods to learn incrementally or learn online is an important feature that makes them available to the real-world problems. Many researchers have devoted to transform the batch SVMs to the incremental ones by adapting incremental learning techniques. The reasons for adopting incremental learning are: (1) incremental learning may be used to keep the memory and computing consumption at an accomplishable level; and (2) incremental learning is wholesome when the useful training examples cannot be

collected before the learning process begins, for example the stream data. According to [3], there are three incremental learning methods: example-incremental learning, class-incremental learning, and attribute-incremental learning. The three methods are to cooperate new examples, new classes, and new attributes to the trained learning system respectively. Syed, Liu, and Sung [4] proposed an incremental SVM (ISVM) and that is one of the earliest attempts to adapt SVM with incremental learning, it re-trains a new SVM by using new examples combined with old SVs. In [5], the old support vectors are more "costly" by adding a constant which may vary with different problems. Laskov et al. [6] proposed an exact online learning algorithm.

For large-scale problems, another approach is to scale down the problem size (down-sampling) and the representatives are used for training. Active learning [7] chooses the representatives by heuristic; Cluster-based SVM (CB-SVM) [8] recursively selects SV clusters along the cluster tree to get better performance; Cluster-SVM [9] first trains an SVM with clusters, then replaces the clusters containing only non-SVs with its representative until each sub-cluster is SVs or non-SVs; Core Vector Machine (CVM) [10] chooses a core set by solving the Minimal Enclosing Ball (MEB) problem; Ball Vector Machine (BVM) [11] places the MEB problem with Enclosing Ball (EB) problem, which is seen as a simplified version of MEB; however, the problem of CVM and BCM is that the algorithms are effective when the size of SVs is small compared to size of the training sample, otherwise the training is very time-consuming; Support Cluster Machine (SCM) [12] exploits a compatible kernel-Probability Product Kernel (PPK) which measures the similarity between clusters and between clusters and vectors.

In this paper, we try to improve SVM with the following three targets: (1) To realize online incremental learning, which is useful especially when the whole data cannot be loaded into memory at the same time or under the online condition; (2) To solve large-scale problem, which is important because the real-world data are often large-scale data. (3) To save number of SVs, the small number of produced SVs generally means small memory requirement and short testing time. We designed an online incremental SVM (OI-SVM) to achieve the three targets. OI-SVM mainly consists of two components: the learning prototypes (LPs) and the learning Support Vectors (LSVs). LPs learns the prototypes and during the learning process continuously adjusts prototypes to the data concept. LSVs are to get a new SVM by combining learned prototypes with trained SVs. OI-SVM can effectively deal with large-scale problems, incremental problems, and online learning problems. It can be applied both in stationary environment and in incremental environment. In the incremental environment, it can handle both the class-incremental problems and the example-incremental problems, and most previous methods have paid less attention to the class-incremental problems.

In the following, Sect. 2 will give a overview of SVM. Section 3 gives the proposed OI-SVM. Section 4 shows some experiments to demonstrate the efficiency of the proposed OI-SVM.

2 Support vector machine (SVM)

For a classification problem, the core idea of SVM is to find the maximum margin hyperplane that separates the labeled training samples. Given data $(x_1, d_1), (x_2, d_2), \ldots, (x_n, d_n)$, $d = 1$ or -1, the hyperplane can be found by solving optimization problem:

$$\max Q(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j d_i d_j x_i^T x_j \tag{1}$$

$$\text{s.t.} \sum_{i=1}^{n} \alpha_i d_i = 0$$

$$0 \le \alpha_i \le C \quad \text{for } i = 1, 2, \ldots, n$$

Examples with non-zero lagrangian multiplier ($\alpha_i \neq 0$) are called Support Vectors (SVs). This method can be extended to nonlinear case by first transforming the input nonlinearly into a high-dimension feature space $\phi : x_i \mapsto \phi(x_i)$ and then making use of the kernel tick

$$K(x_1, x_2) = \phi(x_1)^T \cdot \phi(x_2) \tag{2}$$

Then, calculate SVM with a weighted sum of kernel function outputs, and the corresponding decision bound is

$$f(x) = \text{sign}\left[\sum_{i=1}^{l} \alpha_i d_i K(x, x_i) + b \right] \tag{3}$$

Actually, a key property of SVM is that training an SVM on the SVs alone gives the same result as training on the complete example set.

3 Online incremental SVM (OI-SVM)

In this section, we adapt the traditional SVM into an online incremental version to deal with large-scale problems. To achieve the above-mentioned three targets, we emphasize the necessary representatives of input data–prototypes, the dynamic movement of prototypes adjusts to the data concept, and the role of previous SVs together with the "cost" of old SVs.

In this study, the proposed OI-SVM consists of two components and there is a bridge between them. The two

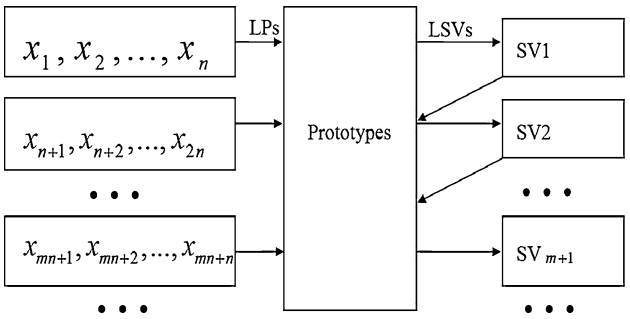

Fig. 1 The *left horizontal arrows* mean that the data are processed by Learning Prototypes (LPs); the *right horizontal arrows* mean that the data are processed by Learning SVs (LSVs); the *declining arrows* mean that the SVs are added to the prototypes, it is called bridge

components are: Learning Prototypes (LPs) and Learning SVs (LSVs). LPs are used to represent the original data and adapt prototypes to the concept of data continuously, the obtained prototypes (representatives) up to now are to represent a local neighbors of the training samples. LSVs are to learn the SVs with the help of learned results of LPs. The bridge between LPs and LSVs is to make the learned data more "costly".

Figure 1 is the structure of OI-SVM. For off-line data, the entire data are partitioned into several parts and each part can fit into memory. For stream data (online data), data flow into the system in online manner. Firstly, training data (x_1, ...x_n, ...) are input to the system. Secondly, LPs learn the input data and generate prototypes to represent the input data. Thirdly, learned prototypes of LPs are input to LSVs to learn SVs. At last, learned SVs are added to the prototypes learned by the next LPs and such SVs and prototypes are used by the next LSVs to generate new SVs.

3.1 Learning prototypes (LPs)

In this part, we will introduce the detail analysis of LPs. LPs are based on the competitive learning, which tries to learn the prototypes to represent a local subset. Given data $(x_1, d_1), (x_2, d_2), \ldots, (x_n, d_n)$, prototypes $(p_1, d_1), (p_2, d_2), \ldots, (p_m, d_m)$ are learned (m is much smaller than n) to fit the density of the training data. Figure 2 is the flowchart of LPs.

There are five modules in Fig. 2: *Insertion, Connection, Update of Threshold, Update of prototypes*, and *Remove*. When a new example (x_i, d_i) is input to LPs, the *insertion* module judges if the new example satisfies the condition of becoming a new prototype. If the answer is "yes," the new example is inserted to the trained system. Else, the other four modules will be executed to learn the information of the new example (x_i, d_i). In the following, we will introduce the five modules.

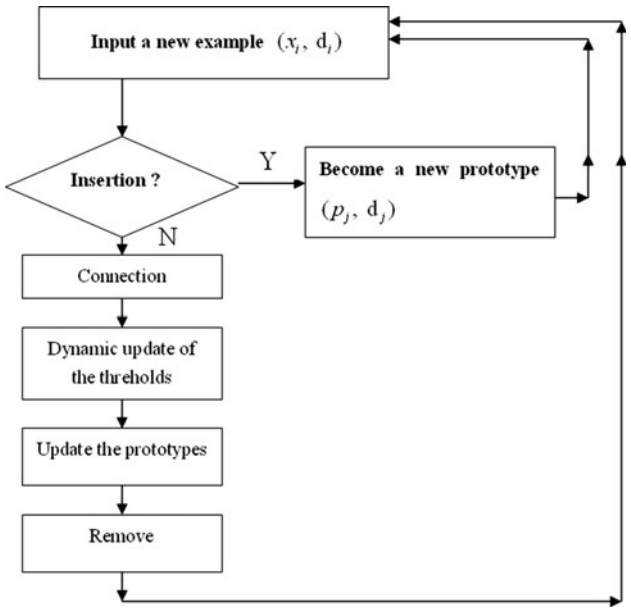

Fig. 2 The flowchart of learning prototypes (LPs)

3.1.1 Insertion

To learn incrementally, the prototype set should grow following the concept of the new data. However, with the prototypes inserted into the set, the insertion of new prototypes must prevent the set from endless growing and over-fitting, and we need to decide when to insert and when to stop. When a new example (x_i, d_i) is input, we insert it to the prototype set directly if the class label d_i is new. If there exists the class d_i, at first, we search the trained prototypes to find the nearest neighbor called "w_{winner}" and the second nearest neighbor called "$w_{\text{runner-up}}$". Then, if the distance between x_i and "w_{winner}" is larger than an automatically adapted threshold of "w_{winner}" or the distance between x_i and "$w_{\text{runner-up}}$" is larger than the threshold of "$w_{\text{runner-up}}$", the new example will be inserted to the prototype set as a new prototype. Here, an automatically adapted threshold is adopted to judge if new prototypes should be inserted. This threshold will be discussed later. The adoption of threshold makes the system generates the number of prototypes properly, and prevents the set from endless growing and over-fitting.

3.1.2 Connection

Topological structure is probed to get the concept of the data. To get reasonable topological structure, the Competitive Hebbian rule is adopted to build connections between prototypes. When two prototypes become the winner and runner-up of an input example, a connection between the winner and the runner-up is set. In incremental or online learning, the prototypes will move slowly and

their neighborhood relations cannot remain forever. The prototypes that have neighborhood relations at earlier stage will not be neighboring at a more advanced stage. So it is necessary to remove connections that have not been renewed recently. Here, the age of a connection is initialized as "0" when it is set, and the age of connections will be incremented during the learning. If the age of one connection is larger than a predefined parameter (*OldAge*), it will be removed.

3.1.3 Update of threshold

For automatically updating of threshold, we adopt two concepts: within-class distance and between-class distance. Within-class distance ($T_{i\text{within}}$) means the average distance between i and other prototypes in its neighborhood and have the same label with i. Between-class distance ($T_{i\text{between}}$) is the distance between i and prototypes whose label is different from i.

$$T_{i_{\text{within}}} = \frac{1}{N_{\text{label}_i}} \sum_{(i,j)\in E \bigwedge \text{label}_i = \text{label}_j} \|\mathbf{w_i} - \mathbf{w_j}\| \tag{4}$$

and

$$T_{i_{\text{between}}} = \min_{(k_1,i)\in E \bigwedge \text{label}_i \neq \text{label}_{k_1}} \|\mathbf{w_i} - \mathbf{w_{k_1}}\| \tag{5}$$

where the Euclidean distance is used as the metric and E is connection set used to store connections between prototypes. In simply words, if prototype a is the neighbor of prototype b, then $(a, b) \in E$; the N_{label_i} is the number of the neighbors of target prototype i. The threshold T_i of prototype i is defined as the largest between-class distance which is less than or equal to $T_{i_{\text{within}}}$.

$$T_i = \max\{\|\mathbf{w_i} - \mathbf{w_k}\|_{\text{label}_k \neq \text{label}_i} \leq T_{i_{\text{within}}} : (k, i) \in E\} \tag{6}$$

3.1.4 Update of prototypes

If a new example cannot become a new prototype, we should update the weight of the winner prototypes to learn the information of the new example. With the help of class labels, we update the weight of prototypes with different strategies according to different class: if $\text{label}_x = \text{label}_{\text{winner}}$, the winner moves toward x and the neighbors of the winner who have different labels with x will move far away from x, else if $\text{label}_x \; != \text{label}_{\text{winner}}$, the winner moves far away from x and the neighbors of the winner who have different labels with x will move toward x. As for the learning rate, we adopt the adaptive learning rate of winner η_1 and the adaptive learning rate of neighbors of the winner η_2 respectively:

$$\eta_1 = \frac{1}{M_{\text{winner}}} \quad \eta_2 = \frac{1}{100M_{\text{winner}}} \tag{7}$$

where M_{winner} is the number of input signals for which this particular prototype has been a winner. This scheme adopted here is to make the position of prototypes more stable when it becomes a winner for more and more input examples. The whole update process makes these prototypes more distinguished from different classes.

3.1.5 Remove

There may be noise in the input data. We will take some efforts to remove the prototypes caused by noise. The noise has strong independent characteristic from the normal data. Here, we adopt the strategy used in [13]: after every several epochs of learning, those prototypes with only one same class neighbor and the winner time is

$$M_{s_i} < 0.5 \sum_{j=1}^{N_G} \frac{M_j}{N_G} \tag{8}$$

will be removed, where N_G is the number of neighbors of a prototype. Also, those prototypes with no topological neighbor will be removed. This strategy works well for removing prototypes caused by noise without adding computing load.

In summary, in this section, an online incremental algorithm for supervised learning is used to represent the training data and get proper prototypes, which reduces the computing consumption of SVM dramatically. The obtained prototypes are necessary to represent the training data up to now.

3.2 Learning SVs (LSVs) and the bridge

One epoch of OI-SVM includes two phase: LPs and LSVs. In one epoch, LPs generate prototypes according to the original data. Then, LSVs will learn the prototypes of LPs to generate SVs. In the next epoch, the LPs will generate new prototype set. The learned SVs of last epoch will be added by the bridge to the new prototype set and get a combined prototype set, then the LSVs will learn the combined prototype set to generate new SVs. This procedure will repeat until whole learning epoches finished.

For incremental learning, usually the new examples have "heavier" effect than the old ones. We need to make the old ones "costly", or in terms of SVM, to make error of the previous learned SVs larger than an error of new examples. SVs are a sufficient representation of the decision hyperplane, but not the all examples. It means that SVs are able to summarize the previous prototypes in a relatively concise way. At the same time, learned prototypes are the representatives of original data and they depict the original data well. Therefore, we add previous learned SVs to new SVM training is able to make the previous data more "costly". That is

$$\phi(w, \xi_i, \xi_i^*) = \frac{1}{2} w^T w + C\left(\sum_{\xi_i \in P} \xi_i + \sum_{\xi_i \in SV} \xi_i^*\right) \quad (9)$$

where P is the set of prototypes and SV is the set contains SVs. Equation (9) is the dual problem of SVM. ξ_i is the slack variable of (x_i, d_i). It is not like the equation in [5]:

$$\phi(w, \xi_i, \xi_i^*) = \frac{1}{2} w^T w + C\left(\sum_{\xi_i \in P} \xi_i + L \sum_{\xi_i \in SV} \xi_i^*\right) \quad (10)$$

where $L = \frac{\#\text{example}}{\#\text{SVs}}$ or $L = 2 * \frac{\#\text{example}}{\#\text{SVs}}$, and # is the examples in the set. Here we need not to use the parameter L. In [5], the use of L can make much larger error on old data when trained new data with old SVs. Here, for OI-SVM, the prototypes learned by LPs can summarize the old data and the prototypes have already make some error on the old data, thus the combination of SVs with prototypes is enough to make the previous data more "costly".

Prototypes produced by LPs may change the original decision bound and generated prototypes are representatives of original data. However, to what extent the obtained prototypes affect the original bound is unknown. Here, we add previously learned SVs into prototypes to train a new SVM, which alleviates the departure of new bound decided by prototypes from original bound.

The training process only preserve prototypes and SVs. We do not need to store all training data. Thus, OI-SVM is able to save plenty of memory. It means that OI-SVM is available for large-scale problem.

3.3 Complete algorithm

With the above discussion, we describe the complete online incremental SVM (OI-SVM) in Algorithm 1.

After processing the last block of off-line data or the last data of online stream data, we get the final SVs that can be used for testing.

In OI-SVM, besides the two parameters in SVM: the regularization parameter C and the parameter of radial-basis-function (RBF) kernel γ, who are often decided by the grid search, there are another two parameters introduced by the LPs component: *OldAge* and λ. According to discussion in [13], the two parameters will be determined by the users, and the generalization ability of the algorithm is not sensitive to the newly introduced parameters.

3.4 Analysis

To handle large-scale problems, keeping the storage and computing consumption at a manageable level is the key issue. Down-sampling is often used to scale down the training set to get much smaller one. When directly solving the large-scale problems is intractable, down-sampling is often used to solve the large-scale problems with limited resources. Meanwhile, incremental learning can also be used to control the storage and computing consumption. Moreover, incremental learning also works when the data cannot be stored in the memory or cannot be scanned repeatedly.

3.4.1 *Down-sampling*

Large training sets pose a great challenge for the storage and computing complexity of a learning algorithm. For SVM, we train large-scale data directly. Because training of standard SVM needs $O(m^3)$ time and $O(m^2)$ space, where m is the training size, the consumed time is so long that sometimes it cannot be tolerated, and let alone the memory consumption. To tackle such large-scale problems, the obvious method is to scale down the number of training examples with down-sampling method. And for down-sampling learning, the problem in essence is how to find the generative representatives to fit the original data.

A commonly used clustering technique is the Threshold Order-Dependent (TOD) algorithm. The main idea of the TOD algorithm is as below. The training examples $(x_1, d_1), (x_2, d_2), \ldots, (x_n, d_n)$ are sequentially processed, and given an example ξ, if the longest distance from ξ to the existing prototypes is larger than the predefined threshold, ξ becomes a new prototype, otherwise ξ belongs to the nearest prototypes. The algorithm is quite attractive because it has a linear complexity and it has good compressing ratio when the predefined threshold is set larger. However, the setting of predefined threshold is quite difficult, when it is set large, all samples will be assigned to one cluster, or if it is set small, each samples will isolate with others. The setting of predefined threshold is problem-dependent and sensitive. For incremental problems, the problem is more difficult because that the earlier setting of predefined threshold may not be suitable to later data. Moreover, the labels of the training examples are not taken into consideration.

In [13, 14], the threshold distance is considered to be dynamic during the learning course. For a prototype x_i, if it becomes the nearest neighbor or the second nearest neighbor of the given example ξ, then the threshold is updated. The concepts of within-cluster distance and between-cluster distance are used. The proposed algorithm in [13, 14] can update the threshold distance without any prior knowledge about the training data. It is designed for the unsupervised learning so that it also does not take labels into consideration.

In this paper, we designed a Learning Prototypes (LPs) method to realize down-sampling for large-scale data. In LPs, we adopt a strategy that dynamically update the threshold distance. The labels of the training examples are

Algorithm 1 Online incremental SVM (OI-SVM)

1: To simulate incremental learning for off-line data, we partition the whole data into n parts (blocks). For online data, the stream data are input to the system directly.

2: Initialize prototype set G, support vector set SV, and edge set E with $\emptyset$.

3: Load one block of data TS into memory. If G is empty, randomly choose two input data from the training set TS, s_1 and s_2, and insert the two data into G, i.e., $G = \{s_1, s_2\}$.

4: Input a new pattern $\xi = (x_i, d_i)$ to the system.

5: Search set G to find the winner s_1 and runner-up s_2 by $s_1 = arg\ min_{c \in G} \|\mathbf{x_i} - \mathbf{w_c}\|$ and $s_2 = argmin_{c \in G \setminus \{s_1\}} \|\mathbf{x_i} - \mathbf{w_c}\|$.

6: **if** $N_{label_{s_i}} < 2 \bigvee \|\mathbf{x_i} - \mathbf{w_{s_1}}\| > T_{s_1} \bigvee \|\mathbf{x_i} - \mathbf{w_{s_2}}\| > T_{s_2}$ **then**

7: Insert ξ into set G. Go to Step(4).

8: **end if**

9: **if** $(s_1, s_2) \notin E$ **then**

10: $E = E \cup \{(s_1, s_2)\}$ and $age_{(s\_1, s\_2)} = 0$.

11: **else**

12: $age_{(s\_1,\ s\_2)} = 0$

13: **end if**

14: **for** s_i in the neighbor area of s_1 **do**

15: Update $age_{(s_1, s_i)} \leftarrow age_{(s_1, s_i)} + 1$

16: **end for**

17: Update winner count: $M_{s_1} \leftarrow M_{s_1} + 1, \eta_1 = \frac{1}{M_{winner}}$, and $\eta_2 = \frac{1}{100 M_{winner}}$.

18: **if** $label_\xi = label_{s\_1}$ **then**

19: Update $\mathbf{w_{s_1}} \leftarrow \mathbf{w_{s_1}} + \eta_1(\boldsymbol{\xi} - \mathbf{w_{s_1}})$

20: **for** s_i in the neighbor area of node s_1 and $label_{s_i} \neq label_\xi$ **do**

21: Update $\mathbf{w_{s_i}} \leftarrow \mathbf{w_{s_i}} - \eta_2(\boldsymbol{\xi} - \mathbf{w_{s_i}})$

22: **end for**

23: **else**

24: Update $\mathbf{w_{s_1}} \leftarrow \mathbf{w_{s_1}} - \eta_1(\xi - \mathbf{w_{s_1}})$

25: **for** s_i in the neighbor area of node s_1 and $label_{s_i} = label_\xi$ **do**

26: Update $\mathbf{w_{s_i}} \leftarrow \mathbf{w_{s_i}} + \eta_2(\xi - \mathbf{w_{s_i}})$

27: **end for**

28: **end if**

29: Delete those edges in set E whose age outstrips the parameter $OldAge$.

30: **if** The number of learned examples i mod $\lambda == 0$. **then**

31: Delete the nodes s_i in set G that have no neighbor node and delete the nodes s_i who has only one neighbor and $M_{s_i} < 0.5 \sum_{j=1}^{N_G} \frac{M_j}{N_G}$.

32: **end if**

33: **if** All the data in the block TS have been processed. **then**

34: Combine the prototype set G and the support vector set SV in new set $Temp$: $Temp = G \bigcup SV$.

35: Train a new SVM with the set $Temp$, get the new support vector set SVs_{new}.

36: Update the old support vector set SV. Delete the old support vectors in SV and add the support vectors in SVs_{new} to SV, i.e., $SV = \emptyset, SV = SVs_{new}$.

37: Go to Step(3) to continue the learning process.

Algorithm 1 continued

38: **else**

39: Go to Step(4) to continue the online learning process.

40: **end if**

41: **if** All the training data have been processed. **then**

42: **return** set SV.

43: **end if**

taken into consideration. The dynamic threshold involves two concepts: within-class distance (the average distance between i and other prototypes in the neighborhood that have same label with i) and between-class distance (the distance between i and prototypes in the neighborhood whose labels are different from i).

In fact, some other supervised classifiers (such as Learning Vector Quantization (LVQ) [15] and its variations, K-Means Classifier (KMC), incremental LVQ [16], and other classifiers [17]) can be used to learn the prototypes of input data and realize down-sampling. We also can use such methods to pre-process original large-scale data and give the representative of the original data. However, how to predefine the number of prototypes is very difficult, and it depends on different tasks. And for large-scale data, the execution of such methods usually needs plenty of training time. Moreover, such methods are not able to deal with incremental learning tasks. The LPs proposed in this paper does not need predefine the number of prototypes, and the number of prototypes is learned automatically by LPs. In advance, as an online learning method, LPs need less training time than some other batch learning methods. Moreover, LPs suit incremental learning very well.

3.4.2 *Online incremental learning*

Online incremental learning is important for real-world problems, especially in such a condition that the data are so large that it cannot be loaded into memory at the same time or the data are not available altogether or cannot be scanned repeatedly, such as stream data. The most important challenge of online incremental learning is when the concept varies between the later learning steps and the earlier learning steps (the so-called concept-drift). Moreover, for class-incremental learning, the problem may be even serious for that there exist new classes unseen by the earlier learning steps.

Essentially, traditional SVM and its variations belong to batch learning. How to adapt traditional SVM to online incremental SVM is very important for SVM suits large-scale problems. Syed, N. et al. presented an incremental SVM (ISVM) in [4]. The main idea of ISVM is to re-train a new SVM with the new data and previous learned SVs

repeatedly. ISVM makes full use of the property of SVM that the decision bound of SVM only depends on SVs, i.e., the SVs can summarize the previous data very well. Based on this property, ISVM can expect to get the incremental learning result same as a batch SVM, and SVs generated during the incremental learning process will end up as SVs in batch learning. In some ideal situations the concept of input data matches the whole training data, ISVM can work well. However, if there exists concept-drift in the online input data, or for class-incremental learning problem, ISVM loses its effectiveness. In such situations, the earlier learning steps are not able to foresee the unknown data or classes, and the earlier SVs have little influence on the later deciding bound. Therefore, only keeping SVs as the representative of the learned data is not able to handle the incremental learning well.

Laskov et al. proposed an online SVM learning algorithm in [6]. The algorithm updates an optimal solution after one training example is added or removed. The major limit of this method is its memory requirement for the reason that the samples near the decision bound must be kept in memory during the entire learning. It means that the algorithm "is unlikely to be scalable beyond tens of thousands examples".

In this paper, we propose an online incremental SVM (OI-SVM) to make SVM suits large-scale problem. OI-SVM consists of two phases: learning prototypes phase (LPs) and learning support vectors (LSVs). LPs are used to do down-sampling for original data, as we mentioned in Sect. 3.1. LSVs are to learn SVs by combining learned SVs with prototypes learned by LPs. With the two phases, OI-SVM is able to provide approximate solutions for online incremental learning tasks, and it suits large-scale problem well.

OI-SVM tries to realize online incremental learning, it means the training time of OI-SVM will be less than some traditional batch learning SVM for large-scale problem. In practice, there are some applications trained "off-line", and longer training time may be tolerated. However, even for such applications the testing time is also important. The speed of testing influences the practicability of a method. The good property of SVM is that when testing, only the SVs are employed. It means that fewer SVs generally mean less testing time. From the experiments in Sect. 4 we know that the proposed OI-SVM can generate fewer SVs than other methods, it means that OI-SVM is effective for practical usage.

With incremental learning and down-sampling learning, OI-SVM saves a lot of memory during the learning process because we do not need to store the whole data set when learning. As the learning results, OI-SVM will output less SVs than some other traditional SVMs and it means that OI-SVM needs less storage of SVs than other methods.

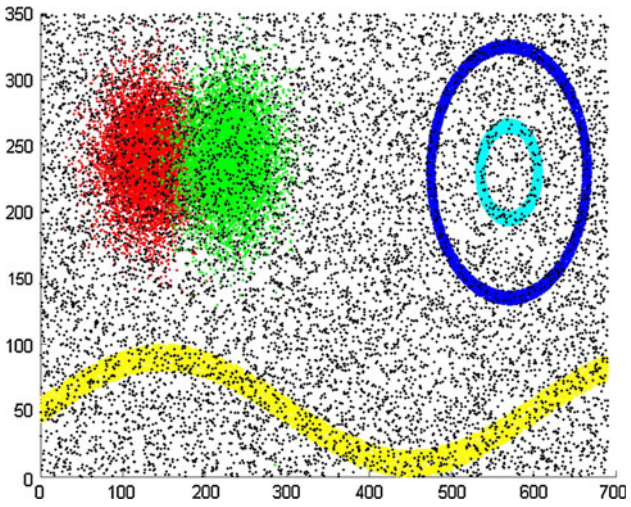

Fig. 3 There are five classes in this figure: two Gaussians, two rings, and one sinusoidal curve. There are 10% noise examples in this figure

Usually, we use compressing ratio to evaluate the performance of SVM algorithms. Compression ratio means the ratio of learned SVs (number of SVs) to whole training data set:

$$r = \frac{s}{t} \tag{11}$$

Here, r means compression ratio, s means number of learned SVs, and t means number of training data.

4 Experiments

In this section, we do experiments on both artificial data and real-world data. The experiments are performed on 3.0 *GHz* Intel Pentium machine with 512 MB RAM. *LIBSVM* (V2.88)[1] [18], *CVM* and *BVM*(*V2.2*)[2] are used to make comparison, and the kernel used in SVM training is RBF kernel. For SVM, it is required to select C and γ in the training phase, we select the optimal parameters by grid search.

4.1 Artificial data

In this section, the experiment is conducted on two-dimensional data, which includes five classes and the topological structure is shown in Fig. 3. We can see that Class 1 and Class 2 overlap and both of them are a 2-dimensional Gaussian distribution. Class 3 and class 4 are two rings having the same center. Class 5 distributes in a

[1] LibSVM are available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.
[2] CVM and BVM are available at http://www.cse.ust.hk/%7Eivor/cvm.htm.

sinusoidal curve. The five classes are marked with different colors, and we randomly select 50,000 patterns (every class in these 5 classes has 8,000 training examples, and 10,000 noise examples are created, and each noise example obeys 2-D uniform distribution and its label is randomly chosen from class 1 to class 5) for training data, and 1,000 patterns (without noise) selected for the testing data.

4.1.1 Batch learning

For batch learning, all the training data are input to the training system once, and processed by the training system. *LibSVM*, *CVM*, and *BVM* are compared with OI-SVM for training time, predicting accuracy, and the number of SVs. For the kernel algorithm, parameters must be chosen. For *LibSVM*, it is required to select C and γ, and we choose C in the rank $\{2^i: -5 \leq i \leq 5\}$ and γ in $\{2^i: -18 \leq i \leq -12\}$. For OI-SVM, we set the parameters *AgeOld* and λ with 50, and C and γ are same with *LibSVM*. Table 1 shows the training time, the predicting accuracy, the number of SVs, and the compression ratio (CR). Figure 4 shows that the final SVs obtained by the four training algorithms.

From the Table 1, we see that OI-SVM gets the same predicting accuracy with other three algorithms, but with less training time, much less SVs, and good compression ratio. Figure 4 shows that OI-SVM removes most of the noise examples, and the other three algorithms seldomly remove these noise examples. From the Fig. 4d, we see that the prototypes trained by LPs have a good description of the original training data. The position of prototypes and the total number of prototypes are determined automatically by LPs. Also, for different classes, LPs may generate different number of prototypes. For the reason that LPs reduce the size of the training set and eliminates much noise, and from the compression ratio we know OI-SVM has a good compression ratio, it saves much training time for later LSVs.

4.1.2 Incremental learning

For incremental learning, we adopt the following scheme: we slit the 50,000 patterns into 5 parts according to their class labels. At first, we use the class 1 and class 2 to train the system, and then add class 3, class 4, and class 5 into the system sequentially. And we use ISVM for comparison. The parameters are the same with the batch learning in Sect. 4.1.1. Table 2 shows the predicting accuracy, the number of SVs, and the compression ratio (CR). Figure 5 shows final SVs obtained by different algorithms.

Comparing Fig. 5a with Fig. 4a, we see that when ISVM learns SVs repeatedly in every steps, the SVs obtained earlier are removed by the training scheme. Therefore it can be stated that ISVM is not able to remember all trained information and it ignores the earlier data. ISVM takes later data more "important" than earlier ones. When tested with earlier data, it is not able to recognize them. And for OI-SVM , the LPs are able to learn new information without ignoring earlier data, and put almost the same "weight" on both new data and old data, and LSVs are able to make the earlier data more "costly".

Compare Table 2 with Table 1, we know that for incremental learning, OI-SVM is able to obtain almost the same predicting accuracy as batch learning.

Table 1 Experiment in batch learning: comparison results for LibSVM, CVM, BVM, and OI-SVM

Algorithm	Training time (s)	Accuracy (%)	#SV	CR (%)
LibSVM	158.63	97.4	16,616	33.232
CVM	5,316.07	97.6	36,594	73.188
BVM	1,027.76	97.4	25,230	50.46
OI-SVM	92.6	98.0	745	1.49

4.2 Real-world data set

In this section, we do experiments for class-incremental learning, example-incremental learning, and large-scale problems. The real data sets used include: (a) *satimage*, (b) *pendigits*,[3] (c) *w3a*, (d) *optidigits*, (e) *shuttle*, (f) *ijcnn*, (g) *webpage*, (h) *sensit vehicle (combined)*, (i) *usps*.[4] In the following experiments, the parameter C is chosen from $\{2^i: -5 \leq i \leq 5\}$ and the parameter γ is in the range of $\{2^j: -5 \leq j \leq 5\}$. The proposed algorithm is conducted under both stationary environment and incremental environment.

4.2.1 Class-incremental learning

In this part, we adopt two well-known multi-class data sets: *satimage* and *pendigits* to test the class-incremental learning. For data set *satimage*, the experiment is conducted as follows: first divide the whole data set into six parts according to their different labels; then choose the two first classes to train a classifier, and add a class once a time to get a new classifier. There are a lot of orders, we choose two from those: the ascending order (*AO*) and the descending order (*DO*) by their number of training examples. The number of training examples of different classes (1, 2, 3, 4, 5, 6) are 1,072, 479, 961, 415, 470, 1,038 (the dimension is 36 and the number of the testing example is 2000). For comparison, we performed the experiments with SVM in batch learning

[3] Pendigits and optidigits are available at http://archive.ics.uci.edu/ml/.

[4] Satimage, w3a, ijcnn, webpage, usps, sensit vehicle (combined), are available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

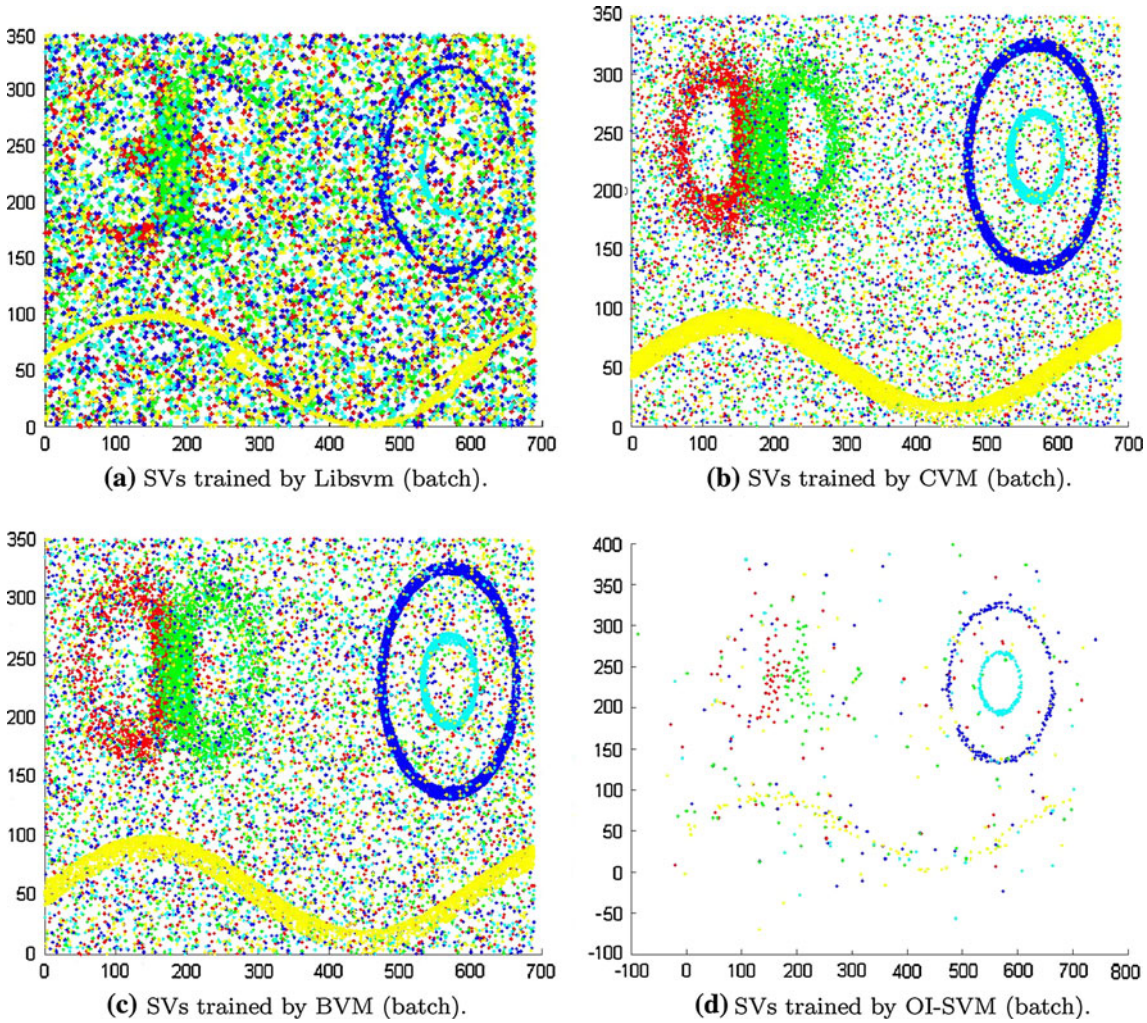

Fig. 4 SVs trained by different algorithms for batch learning

Table 2 Experiment in incremental learning: comparison results for ISVM and OI-SVM

Algorithm	Accuracy (%)	#SV	CR (%)
ISVM	86.5	13,512	27.024
OI-SVM	98.2	1,091	2.182

(LibSVM), Incremental SVM (ISVM), and the proposed OI-SVM. Table 3 shows the test accuracy (with the parameters $OldAge = 2{,}000$ and $\lambda = 400$) and the compression ratio (CR). We also conduct the same experiment on the data set *pendigits* from UCI Machine Learning Repository. It is observed that the data distribute quite evenly, and we arbitrarily choose one order to do the experiment. The testing results for *pendigits* are shown in Table 4.

From Tables 3 and 4 we know that OI-SVM (batch learning) is as accurate as LibSVM (batch), but OI-SVM (batch learning) generates much less SVs than LibSVM; For class-incremental learning, OI-SVM gets much better performance than ISVM for all orders, both in accuracy and number of SVs.

In fact, for class-incremental learning, ISVM only preserve the SVs near the deciding bound in each step. However, for class-incremental learning, the new input training data come from different classes and, if there are only few SVs representing earlier learned data, the influence of such SVs on the later decision bound will be very small, i.e., only preserving SVs cannot handle the class-incremental learning problem well. For OI-SVM, learned prototypes depict the original data well, and we are able to get a global description of the training data. Therefore naturally, OI-SVM works better than ISVM for class-incremental learning.

Actually, when the number of SVs is large enough for its data size, or to be exact the SVs can summarize the data really well in every step, only keeping SVs can also handle the class-incremental learning. From the experiments we

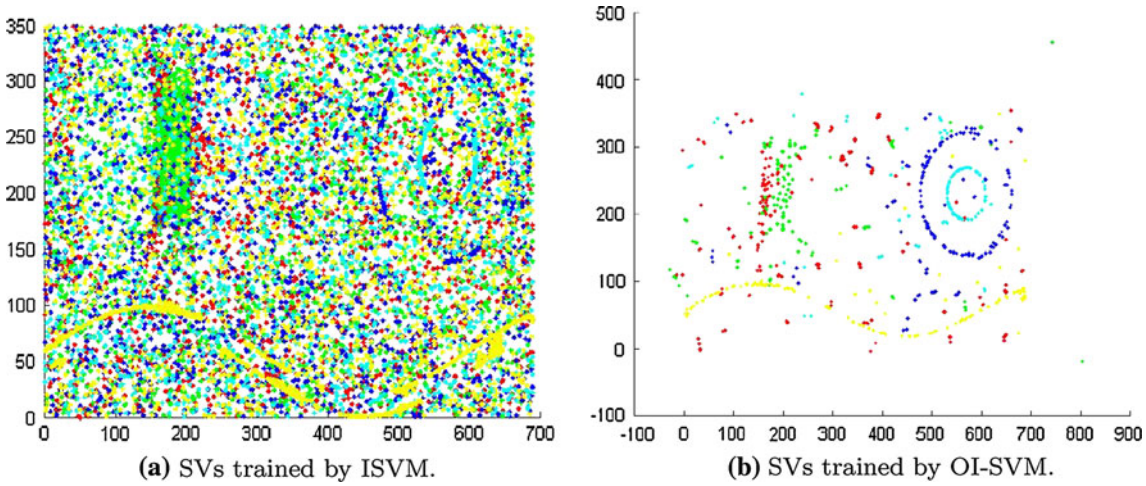

Fig. 5 SVs trained by ISVM and OI-SVM for incremental learning

Table 3 satimage: comparison results for LibSVM, ISVM, and OI-SVM under class-incremental learning environment

Algorithm	Accuracy (%)	#SV	CR (%)
LibSVM (batch)	91.9	2,405	54.23
ISVM (AO)	83.8	1,719	38.76
ISVM (DO)	80.2	1,652	37.25
OI-SVM (batch)	91.55	1,314	29.63
OI-SVM (AO)	91.35	1,635	36.87
OI-SVM (DO)	90.05	1,426	35.15

Table 4 *pendigits*: comparison results for LibSVM, ISVM, and OI-SVM under class-incremental learning environment

Algorithm	Accuracy (%)	#SV	CR (%)
ISVM	94.68	542	7.23
LibSVM (batch)	97.91	1,206	16.09
OI-SVM (incremental)	98.15	687	9.17
OI-SVM (batch)	97.97	431	5.75

know that OI-SVM generates less SVs but obtains better accuracy than ISVM. It means that OI-SVM is able to generate SVs to represent the input data well in every step, and ISVM cannot do that.

Moreover, OI-SVM is able to determine the number of SVs for different classes automatically and keep information about previous prototypes, it means the order of input data imparts small influence for testing accuracy. For example, the accuracy of OI-SVM (AO) for *satimage* is close to OI-SVM (DO), but ISVM (AO) and ISVM (DO) have boarder gap than OI-SVM. However, even with OI-SVM, the order of input data will influence the training time and the number of SVs.

4.2.2 *Example-incremental learning*

In this section, we use data sets *optdigits*, *w3a*, *shuttle*, and *ijcnn* to test example-incremental learning. Such data sets belong to medium data sets: *optdigits* is multi-class and the size of the training set and the testing set is small; the dimension of *w3a* is larger than others but the size of training set is small; *shuttle* and *ijcnn* are large in training size but the dimensions are small. Details of such data sets are shown in Table 5. In this experiment, the data are divided into 10 parts arbitrarily. *LibSVM*, *CVM*, *BVM* are compared with OI-SVM for predicting accuracy and number of SVs. Because *LibSVM*, *CVM*, and *BVM* cannot realize incremental learning, here we only give batch learning results for such methods. We do not give the ISVM results in this experiment, it is because LibSVM has better performance than ISVM [6]. The parameters of OI-SVM for the four data sets are (*OldAge*, λ)= (450, 450), (600, 450), (500, 500), (500, 400), respectively. For *BVM* and *CVM*, the parameters are reported in [10], and we set the recommended parameter $\varepsilon = 10^{-5}$. The results are presented in Table 6.

Take the set *optdigits* as example, from Fig. 6, we find that with the data accumulates, the predicting accuracy

Table 5 Data sets used for experiment

Data sets	Class	Dim	#Train	#Test
Optdigits	10	64	3,823	1,797
w3a	2	300	4,912	44,837
Shuttle	7	9	43,500	14,500
ijcnn	2	22	49,990	91,701
Web	2	300	49,749	14,951
sv(*c*)	3	100	78,823	19,705
usps	2	676	266,079	75,383

Table 6 Example-incremental learning results: the best and the second best performances are highlighted with bold figure

Data	Item	LibSVM (batch)	CVM (batch)	BVM (batch)	OI-SVM (batch)	OI-SVM (incremental)
Optdigits	# of SV	1,594	1,193	2,191	**599**	**987**
	CR (%)	41.70	31.21	57.31	**15.67**	**25.82**
	Accuracy (%)	**98.37**	96.38	96.38	**97.33**	**97.33**
w3a	# of SV	2,270	2,118	**952**	**458**	**1,359**
	CR (%)	46.21	43.12	**19.38**	**9.32**	**27.67**
	Accuracy (%)	**97.85**	97.81	**97.88**	97.40	97.56
Shuttle	# of SV	733	2,112	2,181	**57**	**115**
	CR (%)	1.69	4.86	5.01	**0.13**	**0.26**
	Accuracy (%)	**99.83**	99.74	99.70	99.68	**99.77**
ijcnn	# of SV	**2,547**	9,316	5,271	**1,057**	**4,126**
	CR (%)	**5.10**	18.64	10.54	**2.11**	**8.25**
	Accuracy (%)	**98.54**	98.37	98.38	**98.60**	98.22

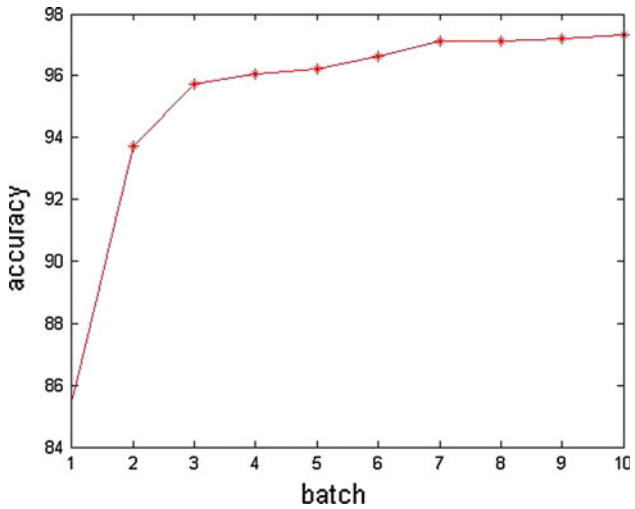

Fig. 6 This figure shows that with the data accumulates, the predicting accuracy is growing

grows with sharp increase firstly and then with slight change in later training phases. When all data have been trained, only 987 SVs have been produced, OI-SVM has very high compressing ratio.

According to Table 6, for batch learning, we find that OI-SVM (batch) is comparable with others for learning accuracy. For all data sets, OI-SVM (batch) generates fewer SVs than others. For incremental learning, even compared with batch learning methods LibSVM, CVM, and BVM, OI-SVM (incremental) also gets comparable accuracy. For number of SVs, OI-SVM gets fewest SVs for "optdigits" and "shuttle". For *w3a*, there are many duplicated examples. During the training of BVM, the duplicated examples are treated as one example. For OI-SVM (incremental), such duplicated examples will be learned repeatedly and it will lead to a little more SVs. For "IJCNN", the number of SVs generated by OI-SVM (incremental) is less than CVM and BVM, but it is greater than LibSVM. For OI-SVM (incremental), the number of SVs generated by the incremental learning is often larger than the batch learning. It is because in the incremental learning, OI-SVM (incremental) works only on a subset of the training examples once a time, and more prototypes are needed to keep more local information and thus get global approximation.

4.2.3 *Large-scale problems*

In this part, the experiments are tested on *webpage, sensit vehicle (combined)*, and *usps*, and details of such data set are shown in Table 5. These data sets are often used as large-scale data to test the performance of learning algorithms. The training sizes of such data sets are very large and the dimensions of examples are also high. During the experiment, we set a limit on the memory—that is 512 MB RAM. The experiment is conducted as follows: for batch learning, the data are input to the system altogether; for incremental learning, the data sets are divided into 10 parts and such parts are input to the system sequentially. Here, parameters of OI-SVM are set for three data sets as: (λ, *OldAge*)=(15,000, 400), (3,000, 500), and (500, 100), respectively, (the parameter γ of the data set *usps* is in the range of $\{2^j: -10 \leq j \leq 0\}$).

From Table 7 we know that, for batch learning, OI-SVM generates much fewer SVs than other methods and obtains comparable accuracy. It means OI-SVM needs less testing time than other methods, and the experiment results also show that the testing speed of OI-SVM is fastest in all four methods. For all data sets, the training speed of OI-SVM is much faster than LibSVM and CVM.

In fact, when the number of support vectors is small for the training data, the training speed of CVM and BVM will

Table 7 Results of experiments for large-scale problem (the best and the second best performances are highlighted with bold figure)

Data	Item	LibSVM (batch)	CVM (batch)	BVM (batch)	OI-SVM (batch)	OI-SVM (incremental)
Web	# of SV	4,652	9,889	3,540	**291**	**464**
	CR (%)	9.35	19.88	7.12	**0.59**	**0.93**
	Accuracy (%)	**99.32**	**99.07**	99.04	98.98	98.95
	Train time (s)	629.65	694.58	**72.38**	**303.38**	420.62
	Test time (s)	26.58	38.46	14.11	**2.24**	**3.70**
sv(c)	# of SV	30,780	40,595	26,087	**6,073**	**7,490**
	CR (%)	39.05	51.50	33.10	**7.70**	**9.50**
	Accuracy (%)	**83.88**	**83.94**	82.49	82.20	82.06
	Train time (s)	4,102.73	55,292.87	11,473.04	**853.57**	**1,203.05**
	Test time (s)	706.39	544.53	338.22	**66.83**	**148.91**
usps	# of SV	2,178	4,094	1,426	**170**	**985**
	CR (%)	0.82	1.54	0.54	**0.06**	**0.37**
	Accuracy (%)	**99.53**	**99.50**	**99.53**	98.96	99.04
	Train time (s)	7,103.17	5,756.19	**1,254.41**	1,392.43	**1,231.72**
	Test time (s)	242.12	379.52	**147.42**	**91.70**	225.64

be fast. However, when the size of SVs is large for its training data, the training speed becomes very slow. In Table 7, for *sv(c)*, the training speed of OI-SVM is much faster than BVM, it is because for BVM, the learned number of SVs is 33% of training data and BVM will be very slow (even less than LibSVM). For *usps*, the training speed of OI-SVM is a little slower than BVM, it is because the number of SVs for BVM is 0.5%. For *web*, the number of SVs for BVM is 7.1%, but BVM works faster than OI-SVM. It is because there are many duplicated examples in *web*.

For incremental learning, OI-SVM almost obtains the similar results of batch learning. It shows that OI-SVM is able to realize incremental learning very well even for large-scale problem.

5 Conclusions

In this paper, we proposed an online incremental SVM (OI-SVM) to deal with large-scale problem. The proposed OI-SVM includes two parts: learning prototypes (LPs) and learning SVs (LSVs). LPs are to learn the proper prototypes to represent the input data, and adjust the prototypes to the concept of new data. LSVs are to learn a new classifier by combining the prototypes representing the data concept with previous learned SVs. OI-SVM can deal with online incremental learning as well as batch learning. For incremental learning, OI-SVM is able to handle both class-incremental learning and example-incremental learning. OI-SVM is not sensitive to the order of input data for incremental learning.

In the experiments, we compared OI-SVM with typical SVM algorithms such as LibSVM, ISVM, CVM, and BVM. The experiments on small-scale data, medium-scale data, and large-scale data show the efficiency of the proposed OI-SVM.

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, BK2011567).

References

1. Vapnik V (1998) Statistical learning theory. Wiley, Chichester
2. Langford J, Li L, Zhang T (2008) Sparse online learning via truncated gradient. In: Advances in neural information processing systems, p 21
3. Zhou ZH, Chen ZQ (2002) Hybrid decisions tree. Knowl Based Syst 15:515–528
4. Syed N, Liu H, Sung K (1999) Incremental learning with support vector machines. In: Proceedings of the workshop on support vector machines at the international joint conference on artificial intelligence (IJCAI-99), Stockholm, Sweden
5. Rüping S (2001) Incremental learning with support vector machines. In: First IEEE international conference on data mining (ICDM'01)
6. Laskov P, Gehl C, Krüger S, Müller K (2006) Incremental support vector learning: analysis implementation and applications. J Mach Learn Res 7:1909–1936
7. Schohn G, Cohn D (2000) Less is more: active learning with support vector machines. In: Proceedings of the international conference on machine learning
8. Yu H, Yang J, Han J (2003) Classifying large data sets using SVMs with hierarchical clusters. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining, pp 306–C331

9. Boley D, Cao D (2004) Training support vector machine using adaptive clustering. In: 4th SIAM international conference on data mining, pp 126–137
10. Tsang IW, Kwok JT, Cheung P-M (2005) Core vector machines: fast SVM training on very large data sets. J Mach Learn Res 6:363–392
11. Tsang IW, Kocsor A, Kwok JT (2007) Simpler core vector machines with enclosing balls. In: 24th International conference on machine learning, pp 911–918
12. Li B, Chi M, Fan J, Xue X (2007) Support cluster machine. In: 24th International conference on machine learning, pp 505–512
13. Shen F, Hasegawa O (2006) An incremental network for on-line unsupervised classification and topology learning. Neural Netw 19:90–106
14. Shen F, Ogura T, Hasgeawa O (2007) An enhanced self-organizing incremental neural network for online unsupervised learning. Neural Netw 20:893–903
15. Kohonen T (1990) Improved versions of learning vector quantization. In: IJCNN90, pp 545–550
16. Xu Y et al (2009) An online incremental learning vector quantization. In: PAKDD 2009, pp 1046–1053
17. Zhang H et al (2006) SVM-KNN: discriminative nearest neighbor classification for visual category recognition. In: CVPR06, pp 2126–2136
18. Chang CC, Lin CJ (2001) LIBSVM: a library for support vector machines. http://www.csie.ntu.edu.tw/cjlin/libsvm