L^3-SVM: a *Life*Long *L*earning Method for SVM

Youlu Xing, Furao Shen*, Chaomin Luo, and Jinxi Zhao

***Abstract*— We propose a lifelong learning method for Support Vector Machine (SVM) training called L^3-SVM. It focuses on the large-scale and endless stream data which are usually the characteristics of lifelong learning. The batch SVM cannot handle large-scale data for the huge space and time requirements and also cannot learn incrementally. In order to solve these two problems, we introduce a Prototype Support Layer (PSL) before SVM training, which is maintained by a Learning Prototype Network (LPN). The LPN learns representative prototypes from the input data in an online way and the PSL records the learned prototypes based on which the SVM is trained. As the size of the prototypes in the PSL is small in comparison with original data, L^3-SVM is very fast. Another challenge of lifelong learning is the "open-ended" environment, i.e. data from novel classes or new distributions may occur during learning, upon which the classifiers should be retrained or updated. Many versions of incremental SVM only retain the Support Vectors after training, losing much potentially useful information of the original data, which leads to a declined performance as new data arrives. The PSL, on the other hand, works as a Long Term Memory (LTM) system. It records the representative prototypes of all previous data. Experiments demonstrate that the SVM trained upon these representative prototypes is as accurate as the state-of-the-art SVM, and much faster and can handle much larger data set than existing methods. Moreover, due to the incremental and self-adaptive properties of LPN, L^3-SVM is able to work efficiently and effectively when novel class data come.**

I. Introduction

Support Vector Machine [1] is one of the most theocratically well motivated and practically successful methods for classification. Traditional SVM adapts a batch method which works as a "*closed*" system: the whole training data set needs to be loaded into memory in advance to run the training process, and the system keeps unchanged after training is done. This leads to many problems: (1) The training of SVM is very, sometimes intolerable, time-consuming and space-consuming, especially in recent years, the size of data set continues to grow at a rapid rate. (2) It cannot be retained with new data set effectively which leads to the system cannot adapt to the *developing* environment, especially the "*open-ended*" environment, i.e. data from novel classes or new distributions occur during learning. The reason is that the system only retains the Support Vectors (SVs) after training is done, the discarded samples may become SVs when some new data are coming, then the performance of the retrained system may be very poor. If the system retains all the previous data (which is a very space-consuming way), the retraining will be very time-consuming as the data accumulates (under large scale problem hypothesis).

In order to solve these two problems, we propose a lifelong learning method for SVM training, called L^3-SVM. It contains a Prototype Support Layer (PSL), which is maintained by a Learning Prototype Network (LPN). The LPN learns the representative prototypes from the input data and the PSL records the learned prototypes. SVM training is conducted on the prototypes in the PSL. Because the size of the prototypes is much smaller, L^3-SVM is very efficient in handling large data both in time and in space. As new data arrives, the PSL will first be updated, then the classifier will be retrained using the prototypes in the latest updated PSL. Because the PSL records the representative prototypes of all previous data and the update of it is extremely fast compared to updating a SVM of the same scale, L^3-SVM can retrain classifiers both efficiently and effectively. That is, L^3-SVM makes the learning system be an "*open-ended*" system.

Briefly, the characteristics of the L^3-SVM are following:

(a) It significantly reduces the computational and spatial costs, especially for large-scale problems, while achieving a comparable classification accuracy.

(b) It can handle new incoming training data efficiently and effectively, the model readily accommodates to lifelong or incremental learning problems.

II. Related Work

A. Data Compression for SVM

Many approaches employ data compression techniques on the training data set to get training set(s) of smaller size to improve the training efficiency on large-scale data set. Clustering-Based SVM (CB-SVM) [2] applies a hierarchical clustering algorithm to obtain a reduced data set used to train SVM. Sun et al. [3] take advantage of k-means based clustering technique to select the data at the boundary of each cluster which they call the "critical data" for SVM training. Some similar approaches are proposed in [4] [5], the clusters are split or shrinked depending on an SVM trained by the centroids of the clusters. Li et al. [6] treat neighboring data as a cluster using a Threshold Order-Dependent (TOD) algorithm. Then they use these clusters standing for the original data set to train the SVM. Romero [7] recommends Leader-SVM which uses the Leader algorithm to cluster the original data set before the SVM is trained with the leader points.

Youlu Xing (youluxing@sina.com), Furao Shen (frshen@nju.edu.cn) and Jinxi Zhao (jxzhao@nju.edu.cn) are with the National Key Laboratory for Novel Software Technology, and Department of Computer Science and Technology at Nanjing University, Nanjing, 210046, P.R.China. Chaomin Luo(luoch@udmercy.edu) is with the Department of Electrical and Computer Engineering at University of Detroit Mercy, Michigan, USA. *Corresponding author is Furao Shen.

978-1-4799-1959-8/15/$31.00

Most of these compression methods need some predefined parameters such as the number of clusters [3], [4], [5] and the similarity threshold [2], [6], [7] which are crucial influence on the clustering result. However, in different kinds of applications, it is very difficult to predefine suitable values for these parameters because they usually depend on different tasks and we usually do not have enough prior knowledge about these parameters. Thus, it will take time to find suitable values for these parameters to get a good data compression result for SVM training.

B. Incremental and Online SVM

Incremental and online learning is important for learning system. It means the learners do not have to start over upon seeing a new piece of data, but can obtain new knowledge and combine with previous one to improve its generalization ability. They also work effectively when data are so large that cannot be loaded into memory at one time or the data are not available altogether, such as stream data.

Syed et al. [8] retrain the classifier with new data and Support Vectors of the current trained SVM. Bordes et al. [9] propose an online SVM which is called LASVM, it is a reorganization of the SMO, the update of the classifier is achieved by alternating two kinds of direction searches, called PROCESS and REPROCESS, when new samples arrive. Orabona et al. [10] propose an online independent SVM, it employs a set of linearly independent observations which they call basis, for each new input sample, if it is linearly independent from the basis, they add this sample to the basis. Singh et al. [11] propose an online SVM, retraining is implemented between a new input sample and the original SVs when the input sample is not correctly classified, if the number of the new SVs is larger than a predefined threshold, they will delete some SVs which do not improve the classifier performance. Zheng et al. [12] propose an online incremental SVM which is also a data compression method for SVM training, it learns the prototypes from the training data in on-line way and uses the learned prototypes to train SVM.

In order to reduce the computational costs and storage costs, most of these incremental and online SVMs [8], [10], [11] discard some input samples which they consider "useless" owing to their reasons during incremental learning. However, the deleted samples may become SVs later, especially when novel classes or new distributions data come. If these discarded samples do not appear in the coming data again, the corresponding concept boundary will be misplaced and the general performance declines.

The L^3-SVM to solve the problems occurred in the *data compression techniques* and *incremental and online SVM* described above is introduced in the following sections.

III. The L^3-SVM

In this section, we first introduce the Learning Prototype Network (LPN) used to obtain prototypes (data compression) from input data. We then provide the batch and incremental methods for training L^3-SVM. We further extend the L^3-SVM to handle novel classes learning problem. Finally, we discuss the space and time complexity of L^3-SVM.

TABLE I
NOTATIONS TO BE USED IN THE LPN.

Notation	Meaning
P	prototype set, used to store prototypes
P_i	prototype i
$\|P\|$	number of prototypes in P
E	connection set, store connections between prototypes
$E_{(P_i,P_j)}$	connection between prototype i and j
$Age_{(P_i,P_j)}$	outdate degree of the connection that connects P_i and P_j
W_{P_i}	weight vector of prototype i
M_{P_i}	local accumulated number of input data of prototype i
N_{P_i}	set of prototypes that directly connected to prototype i, i.e. topological neighbors of P_i
$\|N_{P_i}\|$	number of topological neighbors of prototype i
T_{P_i}	similarity threshold of prototype i

A. Learning Prototype Network

The LPN is an incremental Self-Organizing Map (SOM) [13] type approach. The prototypes and connections between prototypes are created incrementally. The intrinsic structure of LPN enables itself to learn suitable prototypes from data in unsupervised and incremental modes. More advanced than many clustering methods used in SVM training mentioned in Section II.A, *LPN does not need predefined parameters for the number of the prototypes or similarity threshold*. These parameters are self-adjusted adaptively during learning. Moreover, instead of discarding samples that are considered "useless" based on some criterions, LPN updates each prototype to be the local average of the data in the vicinity determined by the aforementioned self-adaptive threshold. In the following, we give the details of the LPN algorithm. Table 1 depicts the definition of notations used in LPN.

The prototype layer of LPN is empty initially. Learning data are fed into the network sequentially, two prototypes are created using the first two input data $\mathbf{x}_1$ and $\mathbf{x}_2$:

$$P_i: \ \{W_{P_i} = \mathbf{x}_i, T_{P_i} = \|\mathbf{x}_1 - \mathbf{x}_2\|, M_{P_i} = 1\}, \ i = 1, 2. \quad (1)$$

i.e. each prototype P_i is associated with a 3-tuple: $\{W_{P_i}, T_{P_i}, M_{P_i}\}$ which represent the weight vector, the similarity threshold and the local accumulated number of input data of P_i, respectively.

For the latter input data $\mathbf{x}$, LPN first conducts competition learning to find the winner prototype P_w and runner-up prototype P_r:

$$P_w = \underset{P_i \in P}{\operatorname{argmin}} \|\mathbf{x} - W_{P_i}\| \quad (2)$$

$$P_r = \underset{P_i \in P - \{P_w\}}{\operatorname{argmin}} \|\mathbf{x} - W_{P_i}\| \quad (3)$$

where $\|\cdot\|$ represents the Euclidean distance. Then LPN decides whether to create a new prototype using the input data or to move the winner prototype with the assistance of the similarity threshold of P_w and P_r. If

$$\|\mathbf{x} - W_{P_w}\| > T_{P_w} \quad \mathbf{or} \quad \|\mathbf{x} - W_{P_r}\| > T_{P_r} \quad (4)$$

, a new prototype P_{new} is created using the input data $\mathbf{x}$:

$$P_{new}: \{W_{P_{new}} = \mathbf{x},\ T_{P_{new}} = +\infty,\ M_{P_{new}} = 1\} \quad (5)$$

else, i.e.

$$\|\mathbf{x} - W_{P_w}\| \leq T_{P_w} \ \ \textbf{and} \ \ \|\mathbf{x} - W_{P_r}\| \leq T_{P_r} \quad (6)$$

the winner prototype P_w will be updated: firstly, the local accumulated number M_{P_w} is added by 1:

$$M_{P_w} = M_{P_w} + 1 \quad (7)$$

then the weight vector of P_w is moved toward the input data $\mathbf{x}$:

$$W_{P_w} = W_{P_w} + \frac{1}{M_{P_w}}(\mathbf{x} - W_{P_w}) \quad (8)$$

After that, self-organizing between prototypes is conducted according to the Hebbian learning rule. Because the distances between $\mathbf{x}$ and P_w, P_r are less than the similarity threshold of P_w, P_r. Prototypes P_w and P_r are activated synchronously. If there is no connection between P_w and P_r, LPN will create a connection between them, and set the outdate degree Age of this connection to 0 to represent the connection as the most recent:

$$E = E \cup E_{(P_w,P_r)}, \quad Age_{(P_w,P_r)} = 0 \quad (9)$$

If a connection already exists, the Age of the connection is set to 0 to strengthen this connection. LPN is an incremental learning method which constantly adjusts itself according to the input environment and the prototypes change their locations gradually during learning. Prototypes that are connected to each other at an early stage might not be near at an advanced stage. The parameter Age of each connection is used to measure the outdated degree of the connection, the movement of P_w will lead to the increase of outdated degree of connections emanating from P_w as:

$$Age_{(P_w,P_j)} = Age_{(P_w,P_j)} + 1, \quad P_j \in N_{P_w} \quad (10)$$

Then LPN will remove the connections whose Age is larger than a predefined threshold Age_{max}.

Similarity threshold T is crucial for clustering. Predefined threshold is not flexible enough for different tasks or dynamic learning environment, meanwhile, it will take time to find an optimum value. In LPN, instead of using a predefined global similarity threshold, each prototype is given a similarity threshold that is adaptively adjusted to the changing environment. Simply stated, the threshold of a prototype T_{P_i} is decided by the prototype-distribution around prototype P_i.

In order to improve the computational efficiency, we only update the similarity threshold of prototype P_w and P_r in practice. Take P_w for example, if P_w has neighbor prototypes, the similarity threshold T_{P_w} is calculated using the maximum distance between P_w and its neighbor prototypes:

$$T_{P_w} = \max_{P_i \in N_{P_w}} \|W_{P_w} - W_{P_i}\| \quad (11)$$

If P_w has no neighbor prototype, the similarity threshold T_w is the distance between P_w and P_r:

$$T_{P_w} = \|W_{P_w} - W_{P_r}\| \quad (12)$$

The similarity threshold of P_r is updated using the same way: If P_r has neighbor prototypes, the similarity threshold T_{P_r} is the maximum distance between P_r and its neighbor prototypes. If P_r has no neighbor prototype, the similarity threshold T_{P_r} is the distance between P_r and P_w.

The original training data may contain noise, some prototypes will be created for the noise data during learning. These prototypes are useless or even harmful. We provide a de-noising function to mitigate the effects of noise. We define noise as low-density distribution data. For the low-density property, the noise prototypes will be isolated and low-frequency winning in competition after a period of learning. Thus, de-noising process is conducted when every λ input data learned using statistical information of the local accumulated number of input data M_{P_i} and the number of neighbors $|N_{P_i}|$ of each prototype P_i. We first calculate the mean value of the local accumulated number of input data of all prototypes as:

$$M_{mean} = \sum_{P_i \in P} M_{P_i}/|P| \quad (13)$$

Then we check the prototypes which have no neighbor or one neighbor, i.e. P_i with $|N_{P_i}| \leq 1$. If

$$|N_{P_i}| = 0 \ \ \textbf{and} \ \ M_{P_i} < M_{mean} \quad (14)$$

or if

$$|N_{P_i}| = 1 \ \ \textbf{and} \ \ M_{P_i} < c \times M_{mean} \quad (15)$$

we mark prototype P_i as a noise prototype and remove it, and connections from P_i are also removed simultaneously. LPN has three parameters: λ, Age_{max} and c, all in relation to noise removal. As described above, λ indicates the frequency of de-noising, and the other two the intensity of it. These parameters are differs parameters presented in Section II.A that define the clustering process.

As a summary, the complete algorithm of the LPN is described in Algorithm 1.

B. The Batch Training Method

To represent notation, consider the classification task given by a training data set $\mathcal{D}$=$\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, each data $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \{-1, +1\}$, let us define $\mathcal{D}^+$ and $\mathcal{D}^-$ as the subsets of the positive and negative data, respectively.

The primary idea of the batch method is as follows: the original training data set is initially decomposed into the positive data part $\mathcal{D}^+$ and the negative data part $\mathcal{D}^-$, before these two parts of data are learned by LPN, respectively. After that, the positive prototype set P^+ and negative prototype set P^- of $\mathcal{D}^+$ and $\mathcal{D}^-$ are acquired. Finally, data set $\mathcal{T} = P^+ \cup P^-$ is used to train SVM.

After training data are learned by the LPN, the positive prototype set P^+ and the negative prototype set P^- are available, therefore data set $\mathcal{T} = P^+ \cup P^-$ for SVM training is obtained. Assume $\mathcal{T} = \{(\mathbf{x}_i, y_i)\}_{i=1}^M$, each data $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i = \{-1, +1\}$. Then the problem of computing a

Algorithm 1 Learning Prototype Network

Input: Training data set $\mathcal{D} = \{\mathbf{x}_i | \mathbf{x}_i \in \mathbb{R}^n\}$, λ, Age_{max}, c.
Output: Prototype set P.

Receive first two input data $\mathbf{x_1}, \mathbf{x_2} \in \mathcal{D}$. Initialize $P = \{P_1, P_2\}$, where $P_i : \{W_{P_i} = \mathbf{x}_i, T_{P_i} = \|\mathbf{x}_1 - \mathbf{x}_2\|, M_{P_i} = 1\}$, $i = 1, 2.$, connection set $E = \emptyset$.
while not end of $\mathcal{D}$ **do**
  Receive input data: $\mathbf{x} \in \mathcal{D}$.
  Find winner: $P_w = \underset{P_i \in P}{\operatorname{argmin}} \|\mathbf{x} - W_{P_i}\|$.
  Find runner-up: $P_r = \underset{P_i \in P - \{P_w\}}{\operatorname{argmin}} \|\mathbf{x} - W_{P_i}\|$.
  if $\|\mathbf{x} - W_{P_w}\| > T_{P_w}$ or $\|\mathbf{x} - W_{P_r}\| > T_{P_r}$ **then**
    Add a new prototype P_{new} to P:
    $P_{new} : \{W_{P_{new}} = \mathbf{x}, T_{P_{new}} = +\infty, M_{P_{new}} = 1\}$
  else
    Update winner prototype P_w:
    $M_{P_w} = M_{P_w} + 1$
    $W_{P_w} = W_{P_w} + 1/M_{P_w}(\mathbf{x} - W_{P_w})$
    if exist connection $E_{(P_w, P_r)}$ **then**
      $Age_{(P_w, P_r)} = 0$
    else
      $E = E \cup E_{(P_w, P_r)}$
      $Age_{(P_w, P_r)} = 0$
    end if
    $Age_{(P_w, P_j)} = Age_{(P_w, P_j)} + 1, \quad P_j \in N_{P_w}$
    if $Age_{(P_w, P_j)} > Age_{max}$ **then**
      Remove connection $E_{(P_w, P_j)}$:
      $E = E - E_{(P_w, P_j)}$
    end if
  end if
  Update the similarity threshold:
  if $|N_{P_w}| \neq 0$ **then**
    $T_{P_w} = \max_{P_i \in N_{P_w}} \|W_{P_w} - W_{P_i}\|$
  else
    $T_{P_w} = \|W_{P_w} - W_{P_r}\|$.
  end if
  if $|N_{P_r}| \neq 0$ **then**
    $T_{P_r} = \max_{P_i \in N_{P_r}} \|W_{P_r} - W_{P_i}\|$
  else
    $T_{P_r} = \|W_{P_w} - W_{P_r}\|$
  end if
  Get the number of input data learned so far as $|\mathcal{D}|$.
  if mod($|\mathcal{D}|, \lambda$)=0 **then**
    Execute prototype pruning:
    $M_{mean} = \sum_{P_i \in P} M_{P_i}/|P|$
    for $i = 1$ to $|P|$ **do**
      if $|N_{P_i}| = 0$ and $M_{P_i} < M_{mean}$ **then**
        Remove prototype P_i:
        $P = P - \{P_i\}$
        $E = E - E_{(P_i, P_j)}, P_j \in N_{P_i}$
      end if
      if $|N_{P_i}| = 1$ and $M_{P_i} < c \times M_{mean}$ **then**
        Remove prototype P_i:
        $P = P - \{P_i\}$
        $E = E - E_{(P_i, P_j)}, P_j \in N_{P_i}$
      end if
    end for
  end if
end while

margin-maximizing boundary function is formulated by the following quadratic programming (QP) problem:

$$\begin{aligned} &minimize : W(\boldsymbol{\alpha}) = -\sum_{i=1}^{M} \alpha_i + \frac{1}{2}\boldsymbol{\alpha}^T \boldsymbol{Q}\boldsymbol{\alpha} \\ &subject\ to : \sum_{i=1}^{M} y_i \alpha_i = 0 \\ &\qquad \forall i : 0 \leq \alpha_i \leq C \end{aligned} \tag{16}$$

where M denotes the number of training data in $\mathcal{T}$. $\boldsymbol{\alpha}$ is a vector of M variables, each component α_i corresponds to a training data $(\mathbf{x}_i, y_i)$. C is the soft margin parameter which controls the influence of the outliers or noise in the training data, and $\boldsymbol{Q}$ is a M-order matrix whose elements are $Q_{ij} = y_i y_j K(x_i, x_j)$, where $K(\cdot)$ is the kernel function.

Instead of calculating the original N-order QP problem, a much small size M-order QP problem is shown in formula (16), which is much time-saving and space-saving. More beneficially, some numerical calculating techniques such as SMO may be utilized to solve formula (16). The complete training algorithm of the Batch L^3-SVM is depicted in Algorithm 2.

Algorithm 2 The Batch L^3-SVM

Input: Training data set $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^n$, $y_i = \{-1, +1\}$.
Output: Decision function f.

Divide training set into the positive and negative subsets: $\mathcal{D} = \mathcal{D}^+ \cup \mathcal{D}^-$.
Calculate positive prototype set: $P^+ = \text{LPN}(\mathcal{D}^+)$.
Calculate negative prototype set: $P^- = \text{LPN}(\mathcal{D}^-)$.
Get compressed training set: $\mathcal{T} = P^+ \cup P^-$.
Calculate decision function: $f = \text{SVMTRAIN}(\mathcal{T})$.

After using Algorithm 2, the decision function f is formulated as:

$$y(\mathbf{x}) = \text{sign}(\sum_{i=1}^{M} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i) + b) \tag{17}$$

where $\mathbf{x}$ is the testing data, $y(\mathbf{x})$ represents the output of the label of testing data $\mathbf{x}$.

C. LifeLong Learning (L^3) for New Block Data

As mentioned in Section II. B, most of the incremental and online SVM discards training samples considered as "useless" during retraining. However, the discarded samples may become SVs later when new data come. In this paper, a point is concluded as follows: *a good data compression is more valuable than a planned data deletion*, because no one knows whether the plan we make can adapt to the changing environment or not.

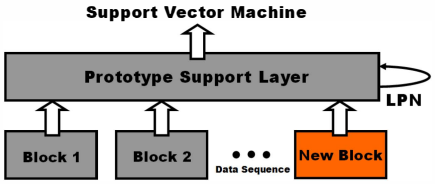


Fig. 1. The Prototype Support Layer works as a Long Term Memory system.

Based on this point of view, we add a **Prototype Support Layer (PSL)** between the learning data and the Support Vector Machine illustrated in Fig. 2. The PSL records the representative prototypes of all the appeared learning data. It functions as a **Long Term Memory (LTM)** system. During learning, new block of data is initially fed to the PSL to extract representative prototypes by LPN before retraining of SVM is conducted using the latest updated prototypes in the PSL. The complete algorithm of lifelong learning for new block data is shown in Algorithm 3.

For the size of the prototype set in the PSL is much smaller than the whole training set, but the expense for retraining is low. On the other hand, the PSL saves representative

Algorithm 3 L^3-SVM for new block data

Premise: Block $1, 2, \ldots$ have been learned by L^3-SVM.
Input: New block data set $\mathcal{N} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where $\mathbf{x}_i \in \mathbb{R}^n$, $y_i = \{-1, +1\}$.
Output: Decision function f.

while not end of $\mathcal{N}$ **do**
  Receive input data: $\{\mathbf{x}, y\} \in \mathcal{N}$.
  if $y = 1$ **then**
    Update P^+ using **Algorithm 1** step 4 to 52.
  else
    Update P^- using **Algorithm 1** step 4 to 52.
  end if
end while
Get prototype set P in the Prototype Support Layer.
Retrain SVM using P: $f = \text{SVMTRAIN}(P)$
Output decision function f.
Waiting for next new block data.

Algorithm 4 L^3-SVM for new classes

Premise: Class $1, 2, \ldots, n-1$ have been learned by L^3-SVM.
Input: New class n data set $\mathcal{N} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where $\mathbf{x}_i \in \mathbb{R}^n$, $y_i = n$.
Output: New SVM classifiers f.

Establish a new prototype storage field $\mathcal{F}$ for class n in the PSL.
Initialize field $\mathcal{F}$ using first two input data of $\mathcal{N}$.
while not end of $\mathcal{N}$ **do**
  Receive input data: $\{\mathbf{x}, y\} \in \mathcal{N}$.
  Update field $\mathcal{F}$ using **Algorithm 1** step 4 to 52.
end while
Get prototypes $P^{\mathcal{F}}$ in $\mathcal{F}$ and prototypes $P^{\mathcal{O}}$ in the original prototype storage field $\mathcal{O}$.
Train new SVM classifiers between class n and the original classes using $P^{\mathcal{F}}$ and $P^{\mathcal{O}}$:
  $f = \text{SVMTRAIN}(P^{\mathcal{F}}, P^{\mathcal{O}})$.
Output new SVM classifiers f.
Waiting for next new class data.

prototypes of the previous training data, the risk that the discarded data may become SVs later is avoided.

D. LifeLong Learning (L^3) for Novel Classes

With the help of the **Prototype Support Layer (PSL)**, L^3-SVM is readily extended to multiclass SVM. As Fig. 3 illustrated, L^3-SVM first learns the prototypes of each class respectively by LPN, then uses the obtained prototypes of each class to train SVM. Users can utilize the prototypes in the PSL to train multiclass SVM with One-Against-One model or One-Against-All model.

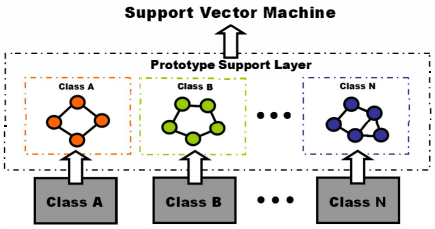


Fig. 2. The structure of multiclass L^3-SVM.

Another issue in lifelong learning is that data from new classes may come during learning. L^3-SVM will establish a new prototype storage field for the new class (assume class n) in the PSL. Then new SVM classifiers between class n and the original classes (One-Against-One model or One-Against-All model) will be trained using the learned prototypes in the PSL. The complete algorithm of lifelong learning for new classes is shown in Algorithm 4.

If new block data and new classes data collect together, users can combine Algorithm 3 and Algorithm 4 conveniently. With these algorithms, L^3-SVM can effectively handle the lifelong learning problem.

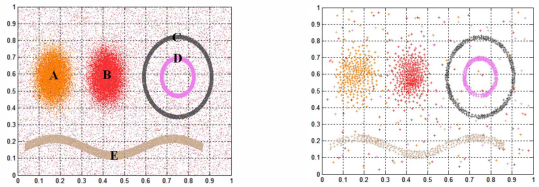
IV. Experiments

In this section, we do experiments on some artificial data sets and real-world data sets to demonstrate the effectiveness of the L^3-SVM[1]. The experiments are performed on 2.50GHz Intel i5-3210M Dual Core CPU with 8.00G RAM.

[1] matlab code of Learning Prototype Network (LPN) can be downloaded at http://cs.nju.edu.cn/rinc/download.html (fastSOINN package)

For comparison, we run some classical and state-of-the-art methods including LibSVM [14] (the latest version V-3.17), ISVM [8], LASVM [9] (the latest version V-1.1) and OI-SVM [12]. Among these methods, the LibSVM [14], CVM [15] and BVM [16] are classified as the batch approaches, whereas the ISVM, LASVM, and OI-SVM are the incremental or online methodologies. In the experiment, we only use the RBF kernel function to train SVM. Parameters C and γ of L^3-SVM are shown in Appendix I.

A. Artificial Data



(a) (b)

Fig. 3. (a) Artificial data set AD5 used in the experiment. (b) The prototypes gained by LPN.

The artificial data set AD5 contains 1.1 million samples from 5 classes: Classes A and B satisfy 2-D Gaussian distribution. Classes C and D are concentric rings distribution. Class E is sinusoidal distribution. Each class has equal numbers of training samples. In order to observe the noise immunity of the algorithms, random noise (10% of useful data) is added to each class. As shown in Fig. 3(a), the noise is distributed over the entire data set. In the training phase, the noise of each class is labeled same as the useful data of each class. In the testing phase, testing samples are randomly generated according to the distributions of the 5 classes without noise. One-Against-One training approach was utilized to compare with other methods because LibSVM uses this approach. In the experiment, the samples are randomly selected from training data set and fed to LPN, we record the learning results every 100 thousands samples learned as stage learning result. The size of the testing set is 10% of the training set at each stage. The parameters of LPN are set as $\lambda = 1000$, $Age_{max} = 1000$, and $c = 0.5$.

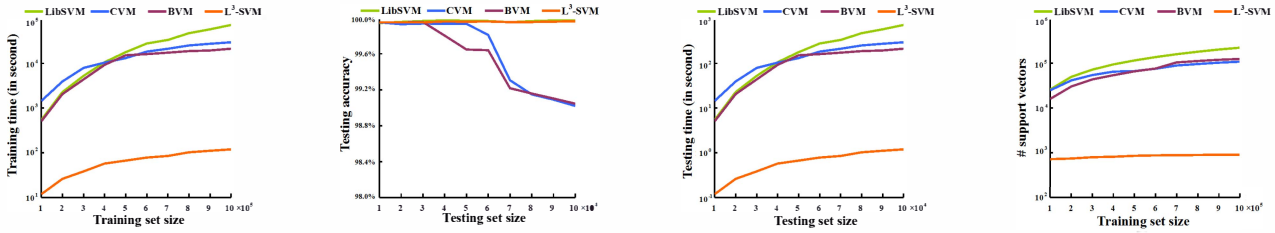


Fig. 4. Results of the batch methods and L^3-SVM on the artificial data set.

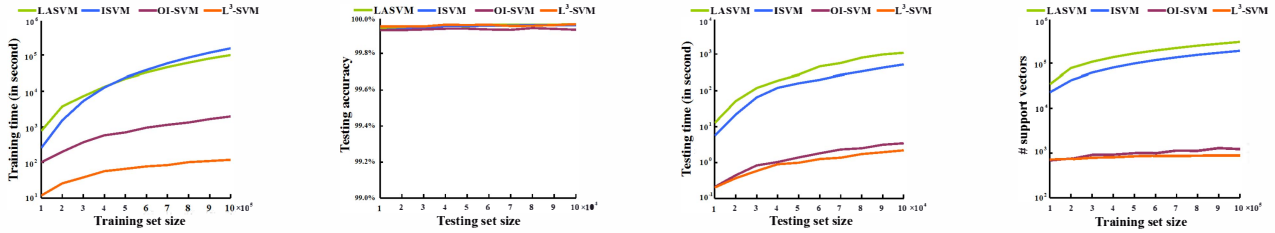


Fig. 5. Results of the incremental methods and L^3-SVM on the artificial data set.

The prototypes gained by LPN are shown in Fig. 3(b). The prototypes fit the original learning set very well. Furthermore, most of the noise is removed. LPN gets 2,185 prototype. The learning data has been effectively compressed. The compression ratio is around 0.20%.

The learning results of the batch methods and L^3-SVM on the artificial data set are shown in Fig. 4. In Fig. 4(a), the training time of L^3-SVM (including learning time of LPN and training time of SVM) is much less than the other methods, meanwhile, the accuracy is comparable with LibSVM and higher than CVM and BVM as shown in Fig. 4(b), the accuracy of CVM and BVM drops may due to the increasing of the amount of noise. Additionally, L^3-SVM generates much fewer support vectors, see Fig. 4(c). Correspondingly, the testing time is much less than these three methods as Fig. 4(d).

The learning results of the incremental methods and L^3-SVM on the artificial data set are shown in Fig. 5. Again, the training time of L^3-SVM (including learning time of LPN and training time of SVM) is much less than the other methods, whereas, the accuracy is comparable with LASVM, ISVM and OI-SVM. Moreover, L^3-SVM generates far fewer support vectors than LASVM and ISVM, therefore, the testing time is much less than these two methods.

B. Real-World Data

In this section, plenty of real-world data sets are adopted, including Letter, MNIST, Sensit Vehicle (combined), Cod-rna, Webspam and MNIST8M[2] to validate our proposed model. The details of the data sets are shown in Table II. This experiment is performed for two environments: (1) in the "closed" environment, all classes of training data are on hand before training. For the LPN, the samples are randomly selected from training data set and fed to LPN. For LibSVM, CVM and BVM, the whole data set need to be loaded into memory before training. If the training set can not be loaded as one batch, LibSVM, CVM and BVM can not work (marked as N/A in the learning result). For the incremental methods ISVM, LASVM, OI-SVM and L^3-SVM, if the training set can not be loaded as one batch, we divide the training set into some small subsets which can be loaded into memory and trained SVM incrementally. (2) in the "open-ended" environment, when the training is fulfilled on the current training set, we will introduce learning samples from new class to the learning system. By taking data set Letter for example, at step 1, we feed letter "a" and "b" to the learning methods. After training is finished, we feed letter "c" to the learning methods at step 2. After letter "c" is learned, letter "d" is fed to the learning methods, etc. This situation aims to test the method whether can deal with new classes data effectively. The parameters of LPN are set as: $\lambda = 1000$, $Age_{max} = 1000$, and $c = 0.1$.

TABLE II
REAL-WORLD DATA SETS USED IN THE EXPERIMENTS.

Dataset	Class	Dimension	# Train	# Test
Letter	26	16	16,000	4,000
MNIST	10	784	60,000	10,000
SVc	3	100	78,823	19,705
Cod-rna	2	8	271,617	59,535
Webspam	2	254	300,000	50,000
MNIST8M	10	784	8,100,000	10,000

1) Closed Environment: The learning results are shown in Table III. It shows that L^3-SVM is much faster than the other methods. Meanwhile, the accuracy of L^3-SVM is very satisfactory, with only a little loss compared to LibSVM. The number of SVs obtained is greatly less than that of LibSVM, CVM, BVM, LASVM and ISVM, resulting in a much faster testing time. This is also very important in some empirical applications. Besides, L^3-SVM can solve efficiently very large scale problems like MNIST8M (about

[2]available at http://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/

TABLE III

LEARNING RESULTS ON THE REAL-WORLD DATA SETS IN THE CLOSED ENVIRONMENT. THE FIRST TERM IN THE TRAINING TIME OF L^3-SVM IS THE LPN LEARNING TIME AND THE SECOND TERM IS SVM TRAINING TIME. THE BEST PERFORMANCE ON EACH DATA SET IS BOLDED.

Dataset		ISVM	LASVM	OI-SVM	CVM	BVM	LibSVM	L^3-SVM
Letter	Training time	25.69	22.35	19.09	10.73	19.87	9.49	**1.70+1.67**
	Accuracy	95.86%	96.90%	95.98%	97.50%	97.00%	**97.78%**	96.05%
	Testing time	1.36	1.62	1.42	2.28	2.68	2.32	**1.20**
	# SVs	4089	5182	4699	7407	8424	7591	**3907**
MNIST	Training time	7911.22	908.42	1721.54	698.67	677.95	610.32	**192.63+60.23**
	Accuracy	97.06%	98.24%	96.85%	98.24%	98.04%	**98.55%**	97.51%
	Testing time	139.49	147.21	77.56	142.78	143.45	174.20	**61.05**
	# SVs	9263	9871	5892	10917	11177	13381	**4987**
SVc	Training time	53820.31	4098.46	1710.06	9842.45	12484.15	27710.48	**154.43+2.41**
	Accuracy	84.07%	85.01%	84.23%	85.74%	85.36%	**86.24%**	85.45%
	Testing time	127.75	61.62	14.97	135.79	188.56	167.82	**9.38**
	# SVs	21144	9871	2352	23801	31488	28864	**1715**
Cod-rna	Training time	123862.99	79465.95	2810.10	6948.69	8641.86	3719.24	**775.2071+10.3589**
	Accuracy	93.04%	94.01%	93.12%	89.93%	87.06%	**94.02%**	93.68%
	Testing time	56.37	74.15	4.37	71.15	71.43	100.23	**2.80**
	# SVs	21599	28362	1466	26566	26906	37413	**1099**
Webspam	Training time	28856.32	9367.06	4587.80	5689.59	7852.22	5543.57	**2903.14+2.77**
	Accuracy	98.47%	99.12%	98.01%	99.12%	99.04%	**99.15%**	98.55%
	Testing time	301.96	325.43	53.72	240.52	290.65	356.43	**34.13**
	# SVs	11839	12767	1990	9515	11589	14303	**1439**
MNIST8M	Training time	$>$ 50 days	1876053.92	604846.32	N/A	N/A	N/A	**87830.38+89.16**
	Accuracy		**99.01%**	94.43%				97.95%
	Testing time		1687.40	108.47				**90.59**
	# SVs		113745	7517				**6722**

1 day) that LibSVM, CVM and BVM can not solve for the memory needed (marked as N/A). For MNIST8M dataset the ISVM have to use more than 50 days[3], LASVM more than 20 days and OI-SVM about 7 days. L^3-SVM can handle large-scale data that will cost intolerable computational cost on other SVMs. Consequently, our method outperforms over most state-of-the-art approaches in terms of computational efficiency and problem scale.

Table IV illustrates the prototype number gained by LPN. The learning data sets have been effectively compressed, especially for large scale data sets. Meanwhile, the compressed data sets guarantee the classification accuracy of SVM (see Table III). Overall, LPN can adapt to various data sets very well, it is very practical and easy-to-use.

TABLE IV

PROTOTYPE NUMBER GAINED BY LPN.

Dataset	# Dataset	# Prototype	Compression Ratio
Letter	16,000	6,394	40%
MNIST	60,000	11,150	18.58%
SVc	78,823	4,262	5.4%
Cod-rna	271,617	4,890	1.8%
Webspam	300,000	5,987	2.0%
MNIST8M	8,100,000	15,471	0.19%

2) Open-ended Environment: Because LibSVM, CVM, BVM and LASVM do not provide the solution to solve the open-ended learning environment problem, we compare L^3-SVM with ISVM and OI-SVM. We perform the experiment on 3 multiclass data sets: SVc, Letter and MNIST.

The learning result on the SVc data set is shown in Table V. The accuracy of ISVM is declining rapidly in the changing environment, from 95.47% to 57.03%. The reason is that when ISVM finishes the training at step 1, it discards the samples which are not SVs, leading to the loss of the potentially useful information in the original data. When new class samples arrives, there is not enough information (samples) to obtain a concept boundary with high accuracy.

TABLE V

LEARNING RESULTS ON THE SVC DATA SET IN THE OPEN-ENDED ENVIRONMENT. THE BEST PERFORMANCE IS BOLDED.

SVc		ISVM	OI-SVM	L^3-SVM
Step 1	Accuracy	95.47%	95.25%	**95.54%**
	Training time	683.72	42.67	**22.13+0.92**
	Testing time	9.32	1.84	**1.32**
	# SVs	3763	585	**498**
Step 2	Accuracy	57.03%	82.06%	**85.45%**
	Training time	1766.15	377.98	**141.81+2.52**
	Testing time	45.84	13.97	**9.56**
	# SVs	7741	2293	**1715**

On the other hand, L^3-SVM retains the prototypes of the training set, so the accuracy of L^3-SVM is guaranteed, actually 85.45% where LibSVM gains 86.24% in the closed environment. Same as in the closed environment, both the training and the testing are much faster than ISVM and OI-SVM.

The learning results on the Letter data set are depicted in Fig. 6. As shown in Fig. 6(a), the accuracy of ISVM and OI-SVM decreases rapidly with arrival of the new classes, approximately 10%. The accuracy of L^3-SVM only decreases slightly after 24 new classes come, about 3%. The cumulative training time of L^3-SVM is less than OI-SVM. Though ISVM achieves less cumulative training time than L^3-SVM, the growth trend of its cumulative training time is much larger than L^3-SVM, (see Fig. 6(b).) Additionally, the testing

[3]the training of ISVM does not finish after 50 days and we stop it, thus we do not give the testing result of ISVM.

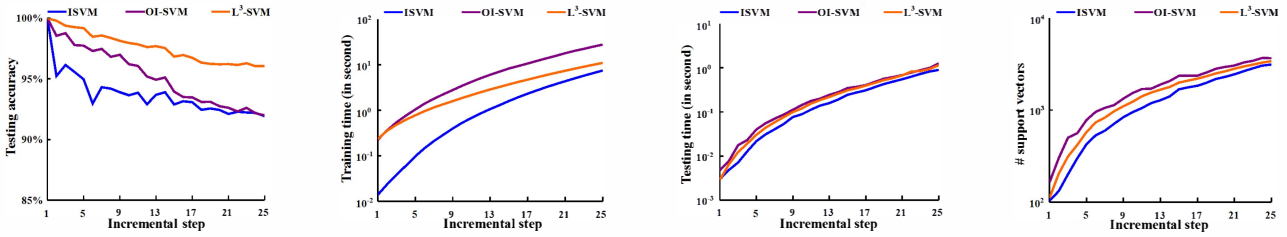


Fig. 6. Learning results of the incremental methods on the Letter data set in the open-ended environment.

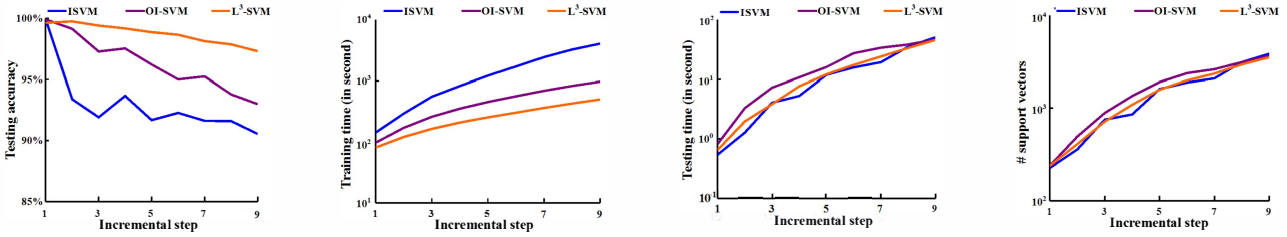


Fig. 7. Learning results of the incremental methods on the MNIST data set in the open-ended environment.

time and SVs number of L^3-SVM is comparable with these two methods shown in Figs. 6(c) and 6(d).

The learning results on the MNIST data set are shown in Fig. 7. Again, the accuracy of ISVM and OI-SVM decreases rapidly with the new classes arrival, 10% and 8%, respectively. The accuracy of the L^3-SVM decreases slightly after 8 new classes arrives, around 2%. The cumulative training time of L^3-SVM is much less than that of OI-SVM and ISVM, and the growth trend of cumulative training time of L^3-SVM is much less than OI-SVM and ISVM (Fig. 7(b)). Besides, the testing time and SVs number of L^3-SVM is comparable with these two methods shown in Figs. 7(c) and 7(d).

V. Conclusion

We propose a lifelong learning method for SVM training to deal with endless stream and large scale data. A Learning Prototype Network (LPN) is introduced to learn the representative prototypes from the learning data. For the open-ended learning environment, we introduce a Prototype Support Layer (PSL) to record the prototypes of the previous data, based on which L^3-SVM can be retrained efficiently and effectively. The experiments demonstrate that the LPN can adapt to various data sets very well, and the L^3-SVM is as accurate as the state-of-the-art SVM, but much faster and can handle much larger data set than those methods. Furthermore, L^3-SVM is able to learn novel classes (or distributions) data much better than the existing incremental or online SVM.

Appendix I

TABLE VI

Parameters C and γ of SVM in L^3-SVM.

	AD5	Letter	MNIST	SVc	Cod-rna	Webspam	MNIST8M
C	100	100	10	10	10000	100000	100
γ	10	0.1	0.3	0.2	0.01	20	0.3

Acknowledgment

This work is supported in part by the National Science Foundation of China under Grant Nos. (61373001, 61375064) and Jiangsu NSF grant (BK20131279).

References

[1] V. Vapnik, *The Nature of Statistical Learning Theory*. New York, USA: Springer-Verlag Press, 1995.

[2] H. Yu, J. Yang and J. Han, "Classifying Large Data Sets Using SVMs with Hierarchical Clusters," *KDD*, 2003, pp. 306-315.

[3] S. Sun, C. L. Tseng, Y. H. Chen, S. C. Chuang and H. C. Fu, "Cluster-based support vector machines in text-independent speaker identification," *IJCNN*, 2004, pp. 729–734.

[4] D. Boley and D. Cao, "Training support vector machine using adaptive clustering," *ICDM*, 2004, pp. 126-137.

[5] J. Yuan, J. Li and B. Zhang, "Learning concepts from large scale imbalanced data sets using support cluster machines," *ACM Multimedia*, 2006, pp. 441–450.

[6] B. Li, M. Chi, J. Fan and X. Xue, "Support cluster machine," *ICML*, 2007, pp. 505–512.

[7] E. Romero, "Using the Leader Algorithm with Support Vector Machines for Large Data Sets," *ICANN*, 2011, pp. 225–232.

[8] N. Syed H. Liu and K. Sung, "Incremental learning with support vector machines," *IJCAI*, 1999, pp. 1–6.

[9] A. Bordes, Seyda Ertekin, Seyda Ertekin and Léon Bottou, "Fast Kernel Classifiers with Online and Active Learning," *Journal of Machine Learning Research*, vol. 6, pp. 1579–1619, Sep. 2005.

[10] F. Orabona, C. Castellini, B. Caputo, L. Jie and G. Sandini, "On-line independent support vector machines," *Pattern Recognition*, vol. 43, no. 4, pp. 1402–1412, 2010.

[11] R. Singh, M. Vatsa, A. Ross and A. Noore, "Biometric classifier update using online learning: A case study in near infrared face verification," *Image and Vision Computing*, vol. 28, no. 7, pp. 1098–1105, 2010.

[12] J. Zheng, F. Shen, H. Fan and J. Zhao, "An online incremental learning support vector machine for large-scale data," *Neural Computing and Applications*, vol. 22, no. 5, pp. 1023–1035, 2013.

[13] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics* vol. 43, no. 1, pp. 59–69, 1982.

[14] C. C. Chang and C. J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, article no. 27, 2013.

[15] I. W. Tsang, J. T. Kwok and P. M. Cheung, "Core vector machines: Fast SVM training on very large data sets," *Journal of Machine Learning Research*, vol. 6, pp. 363–392, Apr. 2005.

[16] I. W. Tsang, A. Kocsor, and J. T. Kwok, "Simpler core vector machines with enclosing balls," *ICML*, 2007, pp. 911–918.