Online Training on a Budget of Support Vector Machines Using Twin Prototypes

Zhuang Wang and Slobodan Vucetic*

Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA

Received 2 November 2009; revised 18 February 2010; accepted 4 March 2010
DOI:10.1002/sam.10075
Published online in Wiley InterScience (www.interscience.wiley.com).

Abstract: This paper proposes twin prototype support vector machine (TVM), a constant space and sublinear time support vector machine (SVM) algorithm for online learning. TVM achieves its favorable scaling by memorizing only a fixed-size data summary in the form of example prototypes and their associated information during training. In addition, TVM guarantees that the optimal SVM solution is maintained on all prototypes at any time. To maximize the accuracy of TVM, prototypes are constructed to approximate the data distribution near the decision boundary. Given a new training example, TVM is updated in three steps. First, the new example is added as a new prototype if it is near the decision boundary. If this happens, to maintain the budget, either the prototype farthest away from the decision boundary is removed or two near prototypes are selected and merged into a single one. Finally, TVM is updated by incremental and decremental techniques to account for the change. Several methods for prototype merging were proposed and experimentally evaluated. TVM algorithms with hinge loss and ramp loss were implemented and thoroughly tested on 12 large datasets. In most cases, the accuracy of low-budget TVMs was comparable with the resource-unconstrained SVMs. Additionally, the TVM accuracy was substantially larger than that of SVM trained on a random sample of the same size. Even larger difference in accuracy was observed when comparing with Forgetron, a popular budgeted kernel perceptron algorithm. As expected, the difference in accuracy between hinge loss and ramp loss TVM was negligible and hinge loss version is preferable due to its lower computational cost. The results illustrate that highly accurate online SVMs could be trained from arbitrary large data streams using devices with severely limited memory budgets. © 2010 Wiley Periodicals, Inc. Statistical Analysis and Data Mining 3: 000–000, 2010

Keywords: support vector machines; online learning; ramp loss; budgeted learning; CCCP

1. INTRODUCTION

An objective of data mining is to develop efficient and accurate algorithms for learning from large quantities of data. Previous research has resulted in many successful learning algorithms that scale linearly or even sublinearly with sample size and dimension, both in runtime and in space. However, linear or even sublinear space scaling is often not sufficient, because it implies an unbounded growth in memory with sample size. This clearly opens another challenge: how to learn from large, or practically infinite, datasets or data streams using memory limited resources.

In this paper, we address the online learning on a budget scenario with the following characteristics: (i) independent and identically distributed training examples are observed sequentially and in a single pass; (ii) the learner can choose to maintain in memory any information about the observed examples; and (iii) the learning should be anytime (i.e., produce an accurate predictor upon unforeseen termination). Evidently, there are many choices as to what the online learner could memorize. The information saved could include a sample of the observed examples, their statistical summary, a prediction model, or any combination of these.

At one end of the spectrum, we have a model-free approach where only a sample or a summary of the observed data is maintained. For example, the reservoir sampling [1] can be used to maintain a random sample from a data stream. Then, training can be carried out offline using the selected examples. Similarly, instead of reservoir sampling, one can decide to use some type of online clustering to better represent data diversity. Although model-free approach is computationally efficient, the caveat is that unsupervised sampling is often not optimal with respect to the learning quality.

At the other end of the spectrum, we have a data-free approach where only the prediction model is saved in memory and updated as new examples are observed. An example of the data-free approach is the perceptron algorithm [2] that converges to the optimal classifier when classes are linearly separable. This algorithm is extremely efficient, both in time and space, but it is applicable only to the very limited class of problems. Most other approaches

Correspondence to: Slobodan Vucetic (vucetic@ist.temple.edu)

are hybrid in that they require memorizing of both data and prediction model.

An example of the hybrid approach is the recursive least squares algorithm [3] that matches learning quality of the memory-unlimited batch solution. In addition to the model weights, the algorithm maintains a data summary in the form of an information or a covariance matrix. Therefore, it is a constant space online algorithm with quadratic scaling with the number of attributes. Development of a similarly successful algorithm for nonlinear regression is still an open problem. Representatives of hybrid approaches in classification are budgeted kernel perceptrons [4–8]. The kernel perceptrons are represented by a subset of observed examples (i.e., support vectors) and their weights, and the budgeted solution is achieved by ensuring that the number of support vectors is bounded. Although there are theoretical guarantees for convergence of budgeted kernel perceptrons under certain assumptions, their performance is often poor on noisy classification problems.

In this paper, we propose a support vector machine (SVM) [9] algorithm for online learning on a budget. SVMs are a popular class of machine learning algorithms that achieve superior accuracy on a number of practical supervised learning problems. Most often, SVM training is formulated as a quadratic programming problem and its solution typically requires quadratic space and cubic time scaling with sample size. Various solutions have been proposed [10–16] to improve the time and space efficiency of SVMs and make them applicable on very large datasets. Although these solutions achieve much better scaling in practical applications, they still have an unbounded growth in memory with the sample size. In addition, they require multiple passes through training data, which is not acceptable in the online learning scenario.

The proposed algorithm, called twin prototype support vector machine (TVM), is designed to handle arbitrarily large data streams while operating under a fixed memory budget. The basic idea of TVM is to upper bound the number of support vectors during the whole learning process. Instead of saving actual examples, the prototype vectors that summarize the local data distribution are maintained. The prototypes and their weights are updated after each newly observed training example. To increase space efficiency, the *twin prototype* is defined by its location and class distribution in its vicinity. The TVM treats prototypes as meta-examples during training.

A critical question about TVM is how to maintain prototypes to maximize the accuracy. One approach is to have them faithfully represent the distribution of the whole data stream. However, this approach could be wasteful because regions far from the decision boundary are less important for SVM. To illustrate this, let us first consider the traditional hinge loss SVM [9]. There, all examples near the decision boundary and all misclassified examples become support vectors. The misclassified examples far from the decision boundary become support vectors as a direct consequence of using the hinge loss, which is a popular choice because it allows casting SVM training as a convex optimization problem. In practice, misclassified examples far from the decision boundary are most often noisy or outlying examples that could lead to reduced SVM accuracy and overly large SVM models. Therefore, ignoring both well-classified and misclassified examples far from the decision boundary is not expected to harm the hinge loss SVM. In the more recently proposed ramp loss SVM [11], where only examples near the decision boundary become the support vectors, this issue is directly addressed. In addition to resulting in smaller predictors, the accuracy of ramp loss SVM tends to have higher accuracy, especially on noisy data. These advantages of ramp loss SVM come at the price of having to solve a more challenging optimization problem.

The discussion above indicates that, for the SVM on a tight budget, positioning prototypes near the decision boundary is better than trying to cover the whole attribute space. In the proposed algorithm, a new example is added as a new prototype if it is near the decision boundary. If this happens, to maintain the budget, the algorithm first looks if some existing prototype is sufficiently far from the decision boundary and removes it. If not, two neighboring prototypes are selected and merged into a single one. Finally, TVM is updated such that the optimal SVM solution is maintained on all the prototypes. To accomplish this, the incremental and decremental updating techniques are applied. The resulting TVM algorithm achieves constant space and linear time scaling and is very appropriate for online learning on a budget from large datasets. By studying incremental and decremental updates for both hinge loss and ramp loss SVM, we intend to show that the particular choice of loss does not have significant influence on accuracy of TVM. As a consequence, we suggest using hinge loss SVM updates due to lower computational cost. We also argue that the resulting TVM predictor resembles the ramp loss SVM.

The paper is organized as follows. In Section 2, we give an overview of the hinge and ramp loss SVMs. In Section 3, we propose twin prototypes for offline SVM training on a budget, and in Section 4 we describe the TVM algorithm. In Sections 5–7, the important components in the TVM algorithm are described. Section 8 provides results of the thorough characterization of TVM on a number of large datasets. The related work is discussed in Section 9.

2. BACKGROUND: HINGE AND RAMP LOSS SVMS

Let us assume we are given a dataset $D = \{(x_i, y_i), i = 1, \ldots, N\}$, where (x_i, y_i) is the ith labeled example, x_i

is an M-dimensional input vector, $y_i \in \{+1, -1\}$ is the associated binary label, and N is the data size. We consider a linear classifier of the form $f(x) = w \cdot \Phi(x) + b$, where w is the weight vector, b is the bias threshold, and Φ is a mapping from the original input space to the feature space. The SVM classifier is trained using the dataset D by optimizing the primal problem

$$\min_{w,b} \frac{1}{2}||w||^2 + C\sum_{i=1}^{N} L[y_i, f(x_i)], \tag{1}$$

where $L[y_i, f(x_i)]$ is some specific loss function and C is the nonnegative user-specified slack parameter that balances the model complexity and loss on training data. It could be shown [11] through differentiating Eq. (1) that the optimal solution w must satisfy

$$w = -C\sum_{i} y_i L'[y_i, f(x_i)]\Phi(x_i). \tag{2}$$

From Eq. (2), it follows that all examples with nonzero loss function derivative $L'[y_i, f(x_i)]$ are used in SVM model. Such examples are called *the support vectors* (SVs).

2.1. Hinge Loss SVMs

The standard loss function for SVM is the hinge loss [Fig. 1(a)] defined as

$$H[y_i, f(x_i)] = \begin{cases} 0, & y_i f(x_i) > 1 \\ 1 - y_i f(x_i), & y_i f(x_i) \le 1 \end{cases}. \tag{3}$$

To optimize the primal problem (1), it is typically transformed to the dual problem as

$$\min_{0\le\alpha_i\le C} : D(\alpha) = \frac{1}{2}\sum_{i,j}\alpha_i Q_{ij}\alpha_j - \sum_i \alpha_i + b\sum_i y_i\alpha_i, \tag{4}$$

where α are the Lagrange multipliers associated with the constraints of the primal problem, $Q_{ij} = y_i y_j k(x_i, x_j)$ are elements of the Gram matrix Q, and k is the positive definite kernel function satisfying $k(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$. The resulting SVM classifier can be conveniently represented using the dual problem solution as

$$f(x) = \sum_{i=1:N} y_i\alpha_i k(x_i, x). \tag{5}$$

The solution of Eq. (4) has to satisfy the Karush–Kuhn–Tucker (KKT) conditions defined as partial derivatives of the objective function $D(\alpha)$,

$$g_i = \frac{\partial D(\alpha)}{\partial \alpha_i} = y_i f(x_i) - 1 \begin{cases} >0;\ \alpha_i = 0 \\ =0;\ 0 < \alpha_i < C \\ <0;\ \alpha_i = C \end{cases}. \tag{6}$$

KKT conditions are the driving force in most practical SVM learning algorithms. Comparing Eq. (2) with Eq. (5), it could be seen that $\alpha_i = -C \cdot H'[y_i, f(x_i)]$ when $y_i f(x_i) \neq 1$. When $y_i f(x_i) \neq 1$, α_i takes values between 0 and C, whereas $H'[y_i, f(x_i)]$ is undefined. This discrepancy can be reconciled by assuming that $H[y_i, f(x_i)]$ is approximated by a smooth function around $y_i f(x_i) = 1$.

2.2. Ramp Loss SVMs

There are two practical issues with hinge loss that are relevant to the design of budgeted online SVM algorithms. The first is scalability. With hinge loss, all misclassified $[y_i f(x_i) < 0]$ and low-confidence correctly classified $[0 < y_i f(x_i) < 1]$ examples become part of the SVM model. As a consequence, as it was shown in Ref. [17], the number of SVs scales linearly with the training size. This scalability problem is particularly pronounced when there is a strong overlap between classes in the feature space. The second problem is that, with hinge loss, noisy and outlying

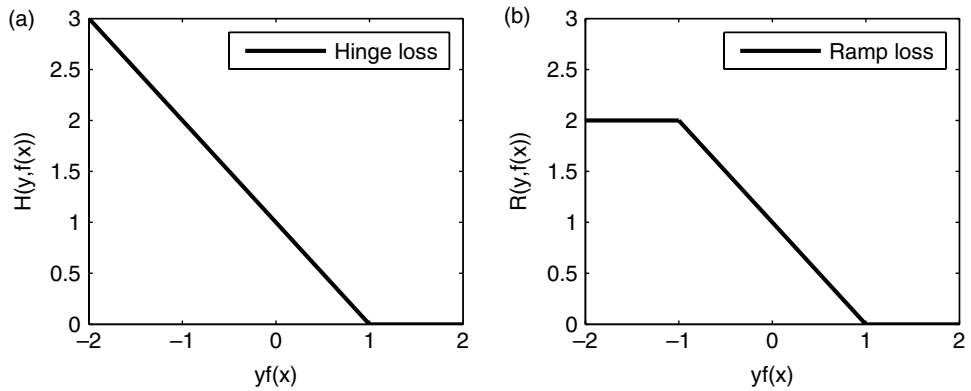

Fig. 1 Hinge loss and ramp loss. (a) Hinge loss. (b) Ramp loss.

examples have large impact on the cost function (1). This can negatively impact accuracy of the resulting SVM. To address the drawbacks of hinge loss, the nonconvex *ramp loss* [Fig. 1(b)]

$$R[y_i, f(x_i)] = \begin{cases} 0, & y_i f(x_i) > 1 \\ 1 - y_i f(x_i), & -1 \le y_i f(x_i) \le 1 \\ 2, & y_i f(x_i) < -1 \end{cases} \tag{7}$$

was introduced in Ref. [11]. Using R in Eq. (1), it could be seen that the examples with $y_i f(x_i) < -1$ do not become SVs. This directly leads to smaller SVM models that are more robust to noise and outliers. Although the ramp loss has very appealing properties, the corresponding optimization problem (1) becomes nonconvex and is harder to solve. In Ref. [11], it was proposed to train ramp loss SVM (SVM^{R}) under the concave–convex procedure (CCCP) framework [18]. The algorithm manages to solve Eq. (1) by iteratively solving a series of hinge loss SVM (SVM^{H}) problems. More details about SVM^{R} and its relation with the proposed TVM algorithm are given in Section 4.2.

3. OFFLINE SVM TRAINING ON A BUDGET USING TWIN PROTOTYPES

Solving primal (1) or dual (4) problem for hinge loss in general requires $O(N^3)$ time and $O(N^2)$ memory. One way to achieve linear time and constant space scaling with data size is to summarize the original dataset with a fixed set of representative prototypes. Let us assume that we are given B prototype attribute vectors $q_1, \ldots, q_B$ and that each training example is represented by its nearest prototype. For the moment, we will not worry about how the prototypes are selected. Using the prototype representation, the original training dataset $D = \{(x_i, y_i), i = 1, \ldots, N\}$ is transformed to $Q(D) = \{[Q(x_i), y_i],\ i = 1, \ldots, N\}$, where Q is the assignment function defined as $Q(x) = \{q_k, k = \arg\min_{j=1:B} ||\text{x- } q_j||\}$. We note that the assignment to the nearest prototype results in the minimization of data distortion, defined as $E[||X - Q(X)||^2]$. Clearly, as the number of prototypes B grows, it is expected that the distortion decreases.

By defining s_j^+ and s_j^- as the numbers of positive and negative examples quantized to prototype q_j, the prototype can be represented by two weighted examples, a positive example $(q_j, +1, s_j^+)$ and a negative example $(q_j, -1, s_j^-)$. We call the pair $tv_j = \{(q_j, +1, s_j^+), (q_j, -1, s_j^-)\}$ the *twin prototype*. Thus, training an SVM on N quantized examples from $Q(D)$ is equivalent to training an SVM on the set of B twin prototypes $TV = \{tv_j, j = 1, \ldots, B\}$,

$$\min_{w,b} \frac{1}{2}||w||^2 + C \sum_{i=1}^{B} \{s_i^+ L[+1, f(q_i)] + s_i^- L[-1, f(q_i)]\}. \tag{8}$$

When L is the hinge loss,[1] after transformation to the dual problem, the KKT conditions become

$$g_i^+ = y_i f(q_i) - 1 \begin{cases} >0; & \alpha_i^+ = 0 \\ =0; & 0 < \alpha_i^+ < s_i^+ C \\ <0; & \alpha_i^+ = s_i^+ C \end{cases}, \tag{9}$$

where $y_i = +1$, such that the feasible range of each Lagrange multiplier depends on s_i^+ weights. The similar conditions apply to g_i^-, with the only difference that $y_i = -1$.

It should be noted that many SVM algorithms, including IDSVM [19] that forms the basis of TVM, can incorporate weights into optimization in a straightforward fashion. Thus, weights of twin vectors s_i^+ and s_i^- do not add to the complexity of SVM training. The resulting budgeted SVM predictor can be expressed as

$$SVM(x) = \sum_{j=1}^{B} (\alpha_j^+ - \alpha_j^-) k(q_j, x) + b. \tag{10}$$

Let us discuss computational costs of the prototype-based budgeted SVM. Because $Q(D)$ can be regarded as a dataset with $2B$ weighted examples, the solution of quadratic programming problem would have $O(B^3)$ time and require $O(B^2)$ memory, which implies constant runtime and constant memory scaling with N. By adding the costs of converting D to $Q(D)$, the runtime becomes $O(B^3) + O(N)$ which represents linear scaling with data size N.

4. TWIN PROTOTYPE SUPPORT VECTOR MACHINE (TVM)

In Section 3, we showed that representing the original dataset D with B prototypes allows SVM training with linear time and constant space scaling with data size. In this section, we explain how we extend this basic idea to develop an online budgeted SVM algorithm. Two basic questions in the design are (i) how to create and update the twin prototype set and (ii) how to efficiently update

[1] As we already mentioned before, SVM^{R} problem can be solved by solving SVM^{H} problem, so we only discuss the case when L is the hinge loss.

the existing SVM solution as new examples are observed. Our idea is to adaptively change the twin prototypes such that they are positioned near the decision boundary and meanwhile use incremental and decremental techniques to maintain the optimal SVM solution on the updated data. The details of the resulting TVM algorithm (outlined in Algorithm 1) for online training on a budget are as follows.

Algorithm 1 TVM

Input: Data stream D, budget B, kernel function k, slack parameter C
Ouput: *TVM* classifier
0. $TVM = 0$; $TV = \emptyset$;
1. **do**
2. receive the new example (x, y);
3. **if** $Accept(x)$ **then**
4. $tv_{\text{new}} = \{(x, y, 1), (x, -y, 0)\}$;
5. $TV = TV + tv_{\text{new}}$;
6. $TVM = Increment(tv_{\text{new}})$;
7. **if** size$(TV) \geq B$ **then**
8. **if** $\max |TVM(q_i)| > m_2$ **then**
9. $k = \arg\max |TVM(q_i)|$;
10. $TV = TV - tv_k$;
11. $TVM = Decrement(tv_k)$;
12. **else**
13. find the best pair tv_i, tv_j to merge;
14. $tv_{\text{new}} = \{(q_{\text{new}}, +1, s^+_{\text{new}}), (q_{\text{new}}, -1, s^-_{\text{new}})\}$;
15. $TV = TV - tv_i - tv_j + tv_{\text{new}}$
16. $TVM = Decrement(\{tv_i, tv_j\})$;
17. $TVM = Increment(tv_{\text{new}})$;
18. **endif**
19. **endif**
20. **endif**
21. **until** the end of the stream

After initializing *TVM* to empty, examples from the stream are read sequentially. For each observed example, we first determine if it should be added to the set of twin prototypes using *Accept* (line 3 in Algorithm 1; details in Algorithm 2). In *Accept*, the first B observed examples are accepted by default. Following examples are accepted if the absolute value of their TVM prediction is below threshold m_1. The basic idea of *Accept* is that examples near the decision boundary are most likely to end up as SVs and that the remaining examples are not likely to have a positive impact on the resulting predictor. By design, *Accept* is consistent with the ramp loss SVM. If $m_1 = 1$ (the default value in TVM), the acceptance region is equivalent to the ramp region in SVM$^{\text{R}}$. Our recent study [20] shows that online training SVM$^{\text{R}}$ by only accepting the examples within the margin can be viewed as a greedy optimization procedure that approximately optimizes the ramp loss cost function (1).

Algorithm 2 Accept

1. **if** size$(TV) < B$ or $|TVM(x)| \leq m_1$ **then**
2. **return** 1
3. **else**
4. **return** 0
5. **endif**

If the example is accepted, it is used to create the new twin prototype tv_{new} (line 4 in Algorithm 1). Then, the twin prototype set *TV* is updated by adding tv_{new} (line 5), and *TVM* model is updated such that it is optimal on the new *TV* (line 6). Details of the TVM updating procedure will be discussed in Section 4.1. If (line 7) the *TV* size (defined as the number of twin prototypes) does not exceed the budget B, then the TVM procedure continues to the next iteration; otherwise the budget maintenance steps are triggered. Budget maintenance recognizes two scenarios. In case when there is a twin prototype that is sufficiently far from the decision boundary ($|TVM(q)|>m_2$; line 8), the procedure finds the farthest prototype (line 9) and removes it from *TV* (line 10) to maintain the budget. The threshold m_2 is selected such that $m_2 > m_1$ [its default value is set to 2, Fig. 2(a)] to create a buffer zone that prevents premature removal of potentially useful twins. Then, TVM model is updated to reflect the change in *TV* (line 11).

In case when all $B + 1$ twin prototypes are within the buffer zone, the procedure selects two neighboring twin prototypes tv_i and tv_j (line 13) and merges them to create the new twin tv_{new} (line 14) that replaces them (line 15). Details of how the two twin prototypes are selected and merged are given in Section 5. Then, *TVM* is updated to reflect the change in *TV* (lines 16 and 17).

4.1. Incremental and Decremental TVM Updates

A core task in *TVM* (Algorithm 1) is updating *TVM* such that it is the optimal solution on *TV* prototype set. Instead of solving the original problem (1) from scratch, it is possible to do it more efficiently using the existing solution. We recognize the *Increment* procedure in which the existing *TVM* is updated upon addition of a prototype to *TV* and the *Decrement* procedure in which the existing *TVM* is updated upon deletion of a prototype from *TV*. In Section 4.1.1, we summarize the previously proposed *Increment* and *Decrement* procedures for SVM$^{\text{H}}$ [19], and in Section 4.1.2 we propose the solutions for SVM$^{\text{R}}$.

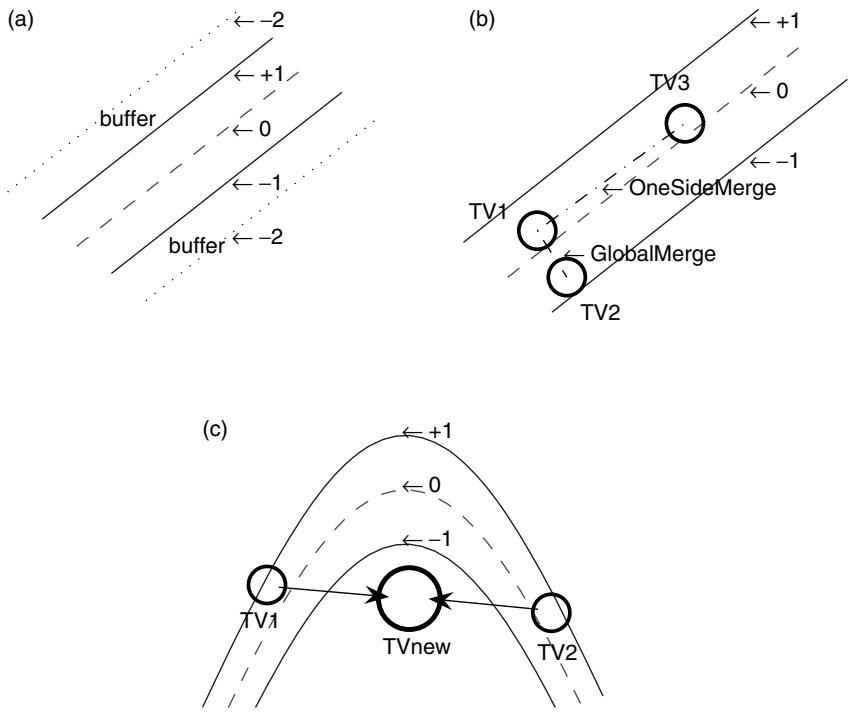

Fig. 2 The illustration of buffer, merging, and rejection. (a) Buffer. (b) Merging. (c) Rejection.

4.1.1. Incremental and decremental learning for hinge loss TVM (TVM^H)

Given an SVM$^{\mathrm{H}}$ model f consisting of N examples and a new example (x_c, y_c), Ref. [19] proposes how to change values of $\alpha_i, i = 1, \ldots, N+1$, and b such that all KKT conditions are maintained on the new data. First, let us denote the SVs that are strictly on the margin [i.e., $y_i f(x_i) = 1$] the *margin* SV (set S), the SVs that violate the margin [i.e., $y_i f(x_i) < 1$] the error SV (set E), and all the other examples the reserve vectors (set R).

If the new example satisfies $y_c f(x_c) > 1$, its α_c value is set to zero, it is directly assigned to the reserve vector set R, and there is no need to update values of α_i's and b. Otherwise, α_c becomes positive, the new example becomes a member of either S or E sets, and all other values of α_i's and b should be updated. By assuming that addition of the new example does not change assignment of the existing N support and reserve vectors, only values of b and α_i's from the margin support vector set S should be modified after inclusion of the new example. The change in g_i value of an example from S set due to a positive value of α_c is expressed differentially

$$\Delta g_i = Q_{ic}\alpha_c + \sum_{j \in S} Q_{ij}\Delta\alpha_j + y_i \Delta b$$
$$0 = y_c\alpha_c + \sum_{j \in S} y_j \Delta\alpha_j, \forall i \in D \cup \{c\}. \qquad (11)$$

Because $\Delta g_i = 0$ for the margin support vector set S with indices $S = \{s_1, \ldots, s_p\}$, where the subscript p is the number of margin support vectors (set S), Eq. (11) could be written in the matrix form as

$$J \cdot \begin{bmatrix} \Delta b \\ \Delta\alpha_{s_1} \\ \vdots \\ \Delta\alpha_{s_p} \end{bmatrix} = - \begin{bmatrix} y_c \\ Q_{s_1 c} \\ \vdots \\ Q_{s_p c} \end{bmatrix} \alpha_c, \qquad (12)$$

where J is symmetric Jacobian matrix

$$J = \begin{bmatrix} 0 & y_{s_1} & \cdots & y_{s_p} \\ y_{s_1} & Q_{s_1 s_1} & \cdots & Q_{s_1 s_p} \\ \vdots & \vdots & \ddots & \vdots \\ y_{s_p} & Q_{s_p s_1} & \cdots & Q_{s_p s_p} \end{bmatrix}. \qquad (13)$$

Matrix J is the enlarged (by one row and column) Gram matrix of the margin support vector examples. By inverting the $(P+1) \times (P+1)$ matrix J, the updated values of α_i's and b can be calculated for any value of α_c.

There are two remaining issues with implementation of the above adiabatic procedure. The first is finding the optimal value of α_c as the value that minimizes the objective function (4). This is accomplished by slowly increasing its value starting from zero as long as Eq. (4) decreases. The second is taking care of the occasional need to reassign examples among the S, E, and R sets due to increase in

α_c. That is accomplished by careful bookkeeping described in detail in Ref. [19]. It is worth mentioning that each bookkeeping step results in incrementing (an example is moved from R or E to S) or decrementing (an example is moved from S to R or E) the Jacobian matrix J by one row and column. After each update of J due to bookkeeping, its inverse should also be updated. The inverse of J is efficiently achieved by rank-one updates.

Similar to the incremental learning, decremental learning (removing any example from the training set) is implemented through adiabatic reversal of incremental learning.

We denote the resulting model updates *Increment*$^{\mathrm{H}}$ and *Decrement*$^{\mathrm{H}}$ and the resulting TVM the hinge loss TVM (TVM$^{\mathrm{H}}$).

4.1.2. *Incremental and decremental learning for ramp loss TVM (TVM^{R})*

In this section, we first introduce the offline algorithm for SVM$^{\mathrm{R}}$ training and then show how to develop ramp loss incremental and decremental procedures for ramp loss TVM (TVM$^{\mathrm{R}}$).

In Ref. [11], it was proposed that the ramp loss objective function (1) can be solved using CCCP. CCCP [18] is an algorithmic framework that optimizes a nonconvex problem by solving a sequence of convex approximations and has no additional parameters to tune.

To optimize Eq. (1) using CCCP, it is decomposed into convex $J_{\mathrm{vex}}(\theta)$ and concave $J_{\mathrm{cave}}(\theta)$ parts, which can be rewritten as [11]

$$\min_{w,b} \left(\underbrace{\frac{1}{2}||w||^2 + C\sum_{i=1}^{N} H[y_i, y_i f(x_i)]}_{J_{\mathrm{vex}}} + \underbrace{C\sum_{i=1}^{N} \{R[y_i, y_i f(x_i)] - H[y_i, y_i f(x_i)]\}}_{J_{\mathrm{cave}}} \right) \quad (14)$$

CCCP runs in iterations. First, $\theta = (w, b)$ is set to an initial guess. In each iteration, $J_{\mathrm{cave}}(\theta)$ is approximated by its tangent $J'_{\mathrm{cave}}(\theta^{\mathrm{old}})\theta$, where θ^{old} indicates θ in the previous iteration and then optimizes the resulting convex function as

$$\theta^{\mathrm{new}} = \arg\min_{\theta}[J_{\mathrm{vex}}(\theta) + J'_{\mathrm{cave}}(\theta^{\mathrm{old}})\theta] = \arg\min_{\theta}\left\{\frac{1}{2}||w||^2 + C\sum\nolimits_{i\in I} H[y_i, y_i f(x_i)]\right\}, \quad (15)$$

where I is the active subset of training examples with margin larger than -1,

$$I = \{\forall i,\ y_i f^{\mathrm{old}}(x_i) \geq -1\}. \quad (16)$$

It is worth noting that the optimization problem (15) is equivalent to training an SVM$^{\mathrm{H}}$ on I. The method in Ref. [11] iteratively retrains SVM$^{\mathrm{H}}$ on the dynamically updated I until I remains unchanged and eventually has all the SVs within the ramp region (i.e., $\forall i, \alpha_i \neq 0, |y_i f(x_i)| \leq 1$).

Clearly, retraining SVM$^{\mathrm{H}}$ from scratch upon every change in I is computationally expensive. Instead, the updates could be performed by using *Increment*$^{\mathrm{H}}$ on each example (or prototype) added to I and *Decrement*$^{\mathrm{H}}$ on each example (or prototype) removed from I. This idea can be directly extended to the online training on a budget. Given the TVM$^{\mathrm{R}}$ trained on the current *TV* set and a new prototype tv_{new} to be added to *TV*, the *Increment*$^{\mathrm{R}}$ procedure can be described by Algorithm 3.

To describe Algorithm 3, we first need to introduce three *TV* subsets: (TV_I) set of all prototypes within the ramp region and TV_U and TV_L as sets of prototypes that were removed from and added to TV_I. If tv_{new} is outside the ramp region, *TVM* is unchanged (line 2 in Algorithm 3). Otherwise tv_{new} is added to the active set TV_I (line 4) and *Increment*$^{\mathrm{H}}$ is applied to update *TVM* (line 5). If in the updated *TVM* some prototypes moved outside the ramp region (line 7) or inside the ramp region (line 8), the algorithm decrements *TVM* with all TV_U prototypes (line 10) and increments *TVM* with all TV_L prototypes (line 11). The CCCP procedure (lines 7–11) is repeated as long as there is a change in TV_I.

The *Decrement*$^{\mathrm{R}}$ procedure can be designed using the same line of reasoning. Let us assume we wish to delete prototype tv_{delete}. We can convert Algorithm 3 to *Decrement*$^{\mathrm{R}}$ algorithm by replacing tv_{new} with tv_{delete} and replacing line 5 with "*TVM* = *Decrement*$^{\mathrm{H}}$(tv_{new});".

4.2. TVM$^{\mathrm{H}}$ Versus TVM$^{\mathrm{R}}$

TVM$^{\mathrm{H}}$ and TVM$^{\mathrm{R}}$ differ in their *Increment* and *Decrement* procedures. As seen in Section 4.1.2, the ramp-based versions are repeatedly calling the hinge-based versions and are therefore more costly. Interestingly, both TVM$^{\mathrm{H}}$ and TVM$^{\mathrm{R}}$ are expected to result in a very similar predictor. It is easy to see why. After accepting the first B examples as B prototypes, TVM accepts only prototypes within the ramp region ($|TVM(q)| < m_1 = 1$). Meanwhile, to maintain the budget, it removes prototypes outside the buffer region ($|TVM(q)| > m_2 = 2$). As a result, most of the prototypes will end up within the ramp region. A tiny fraction of prototypes is expected to remain within the buffer region

Algorithm 3 Increment$^{\mathrm{R}}$

Input: TVM, tv_{new}
Output: TVM
/*
TV: the prototype set in TVM;
TV_I: the active set;
TV_U: the set includes the twin prototypes to be unlearned by CCCP;
TV_L: the set includes the twin prototypes to be learned by CCCP. */
1. **if** $|TVM^{\mathrm{R}}(q_{new})| \geq 1$
2. **break**;
3. **else**
4. $TV_I = \{tv_i, |TVM(q_i)| < 1\}$
5. $TVM = Increment^{\mathrm{H}}(tv_{new})$;
6. **do**
7. $TV_U = \{tv_i \in \mathrm{TV}_I \wedge |TVM(q_i)| > 1\}$;[2]
8. $TV_L = \{tv_i \notin \mathrm{TV}_I \wedge |TVM(q_i)| \leq 1\}$;
9. $TV_I = TV_I - TV_U + TV_L$;
10. $TVM = Decrement^{\mathrm{H}}(TV_U)$; /* sequentially unlearn all elements in TV_U */
11. $TVM = Increment^{\mathrm{H}}(TV_L)$; /* sequentially learn all elements in TV_L */
12. **until** TV_I is unchanged
13. **endif**

($m_1 < |TVM(q)| < m_2$). This is because prototype merging, described in detail in the following section, is drawing all prototypes toward the ramp region. In the extreme case when all prototypes are within the ramp region, ramp loss and hinge loss TVM are identical. Given the small number of possible prototypes within the buffer region, the difference between TVM$^{\mathrm{H}}$ and TVM$^{\mathrm{R}}$ is expected to be very small.

Considering the computational costs, TVM$^{\mathrm{H}}$ is our default TVM algorithm. The experimental comparison between TVM$^{\mathrm{H}}$ and TVM$^{\mathrm{R}}$ shows the success of this approximation and will be illustrated in Section 8.4.

5. MERGING METHODS

In this section, we discuss several methods for selecting the best twin prototypes for merging in Algorithm 1.

5.1. Input Versus Feature Space Merging

5.1.1. Merging in input space

In case when all B twin prototypes are within the buffer zone, the procedure selects two prototypes tv_i and tv_j and merges them to create a new prototype tv_{new} that replaces them. Given tv_i and tv_j, one approach to calculate q_m is to minimize distortion in the input space

$$q_{\mathrm{new}} = \arg\min_q s_i||q_i - q||^2 + s_j||q_j - q||^2, \tag{17}$$

where $s_i = s_i^+ + s_i^-$ and $s_j = s_j^+ + s_j^-$. It follows that the optimal q_{new} is

$$q_{\mathrm{new}} = \frac{s_i q_i + s_j q_j}{s_i + s_j} \tag{18}$$

and that the resulting minimal distortion, denoted as d_{ij} is

$$d_{ij} = \frac{s_i s_j ||q_i - q_j||^2}{s_i + s_j}. \tag{19}$$

The count of positive (or negative) examples s^+_{new} (or s^-_{new}) in tv_{new} is calculated by adding the positive (or negative) counts of the two merged prototypes.

The best pair tv_i and tv_j of twin prototypes to merge is the one that has minimal d_{ij} value. Although it seems that merging in input space requires $O(B^2)$ time and memory, it is evident that, with some bookkeeping from the previous iterations, both costs can be reduced to $O(B)$ in expectation. Specifically, for each twin prototype tv_i, we memorize index k of its best merging neighbor, defined as $k = \arg\min_j d_{ij}$, together with distortion d_{ik}. After that, we calculate distortions between the newly created twin prototype and the existing twin prototypes and use that to modify the memorized indices and distortions.

[2]For ease of presentation, here we assume positive and negative counts of each twin prototype are nonzero.

5.1.2. *Merging in feature space*

An alternative to merging in input space is merging in feature space. Given tv_i and tv_j, the goal is to find q_{new} that minimizes feature space distortion defined as

$$q_{\text{new}} = \arg\min_q s_i||\Phi(q_i) - \Phi(q)||^2 + s_j||\Phi(q_j) - \Phi(q)||^2. \tag{20}$$

Similar to results of Eq. (18), the optimal $\Phi(q_{\text{new}})$ is

$$\Phi(q_{\text{new}}) = \frac{s_i\Phi(q_i) + s_j\Phi(q_j)}{s_i + s_j}. \tag{21}$$

The challenge with feature space merging is that the pre-image of $\Phi(q_m)$ may not exist (as is the case with RBF kernels). As proposed in Ref. [21], this issue can be addressed by finding the approximate solution q^*_{new} by minimizing

$$\min_{q^*_{\text{new}}} ||\Phi(q^*_{\text{new}}) - \Phi(q_{\text{new}})||^2. \tag{22}$$

The general solution of Eq. (22) is hard to achieve because it requires nonlinear optimization that depends on the choice of kernel. In Ref. [21], an algorithm was developed to solve Eq. (22) for RBF and polynomial kernels. Interestingly, in both cases, q^*_{new} lies on the line connecting q_i and q_j.

Because of the increased computational costs of feature space merging, TVM uses input space merging as the default choice.

5.2. Global Versus One-Sided Merging

Without an additional constraint, TVM allows merging of twin prototypes across the decision boundary. Potential issue with such merging is loss of margin support from the original prototypes and creation of a new prototype near the boundary with nearly equal-sized positive and negative twins. This can be harmful to the quality of TVM. In *One-Sided* merging, the twin prototypes are separated into those in the positive region, $TVM(q_i) > 0$, and the negative region, $TVM(q_i) < 0$. Only pairs of twin prototypes from the same region are considered for merging. As an illustration, although merging of tv_1 and tv_2 from Fig. 2(b) would result in the smallest distortion, *One-Sided* would merge tv_1 and tv_3. *One-Sided* merging is the default choice in TVM.

5.3. Merging Rejection

Merging can be inappropriate in the regions of the input space where TVM decision boundary is highly nonlinear. For example, merging of twin prototypes tv_1 and tv_2 from Fig. 2(c) that are at the positive margin results in tv_{new} that is well beyond the negative margin. We note that this behavior is common to both input and feature space merging. To avoid negative effects of such merging, we compare value of $TVM(q_{\text{new}})$ with its estimated value under the linear assumption

$$\hat{TVM}(q_{\text{new}}) = \frac{s_i TVM(q_i) + s_j TVM(q_j)}{s_i + s_j}. \tag{23}$$

Rejection accepts the merging if $TVM(q_{\text{new}})$ is near its estimated value

$$(1-\eta)\hat{TVM}(q_{\text{new}}) < TVM(q_{\text{new}}) < (1+\eta)\hat{TVM}(q_{\text{new}}), \tag{24}$$

where $\eta = 0.2$ is used as the default value. If the selected pair of twin prototypes fails the test, *Rejection* considers the next best merging candidates. If they pass the test, the merging is executed. Otherwise, *Update* concludes that merging cannot be successfully performed. In this case, the new twin prototype is decrementally unlearned by TVM and removed from the twin prototype set *TV*.

6. CONTROL OF C

In online learning on a budget, an additional challenge is the choice of the slack parameter C that controls the trade-off between model complexity and hinge/ramp loss. If C is fixed, the influence of hinge/ramp loss would grow with data stream size due to increase in weights s_j^+ and s_j^- of twin prototypes. Compounded with the bounded number of twin prototypes in TVM, which constrains its representational power (TVM could span at most a B-dimensional manifold in the feature space), fixed C would lead to overfitting.

To address this issue, we dynamically adjust slack parameter C such that the product $C^* = C\sum_j (s_j^+ + s_j^-)$ is kept constant during the online learning process. Adjusting TVM with respect to a change in C could be performed efficiently using path regularization techniques [22]. In TVM, we use regularization parameter permutation [23] which is a method implemented in IDSVM algorithm.

7. COSTS OF TVM ONLINE TRAINING

The memory requirement of TVM is constant with sample size and scales as $O(B^2)$ with the budget B due to the maintenance of Jacobian matrix J in Eq. (13). For runtime cost, the *Accept* needs to provide TVM prediction for a new example which is $O(B)$. In the worst case, merging of two prototypes is performed for budget maintenance. Finding the best pair for merging requires $O(B)$

expected runtime subject to appropriate bookkeeping from previous iterations. Dynamically adjusting C takes $O(B)$ time. Finally, the runtime of *Increment*$^{\mathrm{H}}$ *and Decrement*$^{\mathrm{H}}$ is $O(B^2)$ which is dominated by the recursive method to update inverse matrix. The total runtime for TVM$^{\mathrm{H}}$ is therefore $O(NB + nB^2)$, where n is the number of examples among the N observed examples which are selected by *Accept*. Observe that ratio n/N is initially close to 1 and that it decreases with N when threshold m_1 is at its default value 1. This is because the margin decreases with N which causes *Accept* to become more and more selective. Because nB^2 dominates the runtime and $n = O(N)$, the total runtime appears sublinear in N. The runtime of TVM$^{\mathrm{R}}$ is larger due to increased cost of *Increment*$^{\mathrm{R}}$ and *Decrement*$^{\mathrm{R}}$.

8. EXPERIMENTAL RESULTS

In this section, we present results of detailed evaluation of TVM on a number of benchmark datasets.

8.1. Datasets

Properties of 12 benchmark datasets[3] for binary classification are summarized in Table 1. The multiclass datasets were converted to two-class sets as follows. For the digit datasets *Pendigits* and *USPS*, we converted the original ten-class problems to binary by representing digits 1, 2, 4, 5, 7 (nonround digits) as negative class and digits 3, 6, 8, 9, 0 (round digits) as positive class. For *Letter* dataset, negative class was created from the first 13 letters of the alphabet and positive class from the remaining 13. The seven-class *Shuttle* dataset was converted to binary data by representing class 1 as negative and the remaining ones as positive. Class 1 in the three-class *Waveform* was treated as negative and the remaining two as positive. For *Covertype* data, the class 2 was treated as positive and the remaining six classes as negative. *Adult, Banana, Gauss*, and *IJCNN* were originally two-class datasets. *Gauss* data was generated as a mixture of 2 two-dimensional Gaussians: one class is from $N[(0,0), I]$ and the other is from $N[(2,0), 4I]$. *Checkerboard* data was generated as a uniformly distributed two-dimensional 4×4 checkerboard with alternating class assignments (Fig. 3). *Checkerboard* is a noise-free dataset in the sense that each box consists exclusively of examples from a single class. *N-Checkerboard* is a noisy version of *Checkerboard* where class assignment was switched for 15% of the randomly selected examples. For both datasets, we used the noise-free *Checkerboard* as the test set. In this way, the highest reachable accuracy for both *Checkerboard* and *N-Checkerboard* was 100%. Attributes in all datasets were scaled to mean zero and standard deviation 1.

[3] *Adult, Covertype, IJCNN, Letter, Shuttle* and *USPS* are available at http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/, *Banana* is available at http://ida.first.fhg.de/projects/bench/benchmarks.htm, and *Pendigits* and *Waveform* are available at http://archive.ics.uci.edu/ml/datasets.html.

Table 1. Dataset summaries.

Datasets	Training size	Testing size	Dim
Adult	21 048	9 114	123
Banana	4 300	1 000	2
Checkerboard	100 000	5 000	2
N-Checkerboard	100 000	5 000	2
Covertype	100 000	10 000	54
Gauss	100 000	5 000	2
IJCNN	49 990	91 701	22
Letter	16 000	4 000	16
Pendigits	7 494	3 498	16
Shuttle	42 603	14 167	9
USPS	7 291	2 007	256
Waveform	100 000	5 000	21

8.2. Evaluation Procedure

We used four SVM algorithms, LIBSVM [24], SVM$^{\mathrm{R}}$, IDSVM [19], and TVM, and one kernel perceptron algorithm (Forgetron [6]) in the experiments. Because IDSVM is a memory-unconstrained online algorithm, it serves as an upper bound on accuracy achievable by TVM. LIBSVM, one of the most successful offline hinge loss SVM algorithms, is both highly accurate and computationally efficient which is served for benchmarking of other algorithms. SVM$^{\mathrm{R}}$ denotes our implementation for offline ramp loss SVM in Ref. [11], where LIBSVM serves as the core procedure within CCCP. SVM$^{\mathrm{R}}$ is useful because of the observed similarities between TVM and ramp loss SVM. In addition, we also used the self-tuned Forgetron algorithm [6], a popular kernel perceptron algorithm for online learning on a budget. For TVM experiments, we evaluated five different budgets, $B = 20, 50, 100, 200, 500$. The default TVM parameters $m_1 = 1, m_2 = 2$, with *InputSpace* and *One-Sided* merging, and *Rejection*. TVM$^{\mathrm{H}}$ was used in all experiments, unless if specified otherwise. Training examples were ordered randomly. Additionally, we also trained IDSVM on a random sample of size $B = 100$ from training data and denoted it as *Random*. It is a representative of the model-free online learning approach and serves as a lower bound on the accuracy achievable by TVM with $B = 100$.

We performed three set of experiments on 12 datasets with three different kernels:

- linear kernel $k(x, y) = x \cdot y$,
- RBF kernel $k(x, y) = \exp(-||x - y||^2/2\delta)$,
- polynomial kernel $k(x, y) = (x \cdot y + 1)^d$.

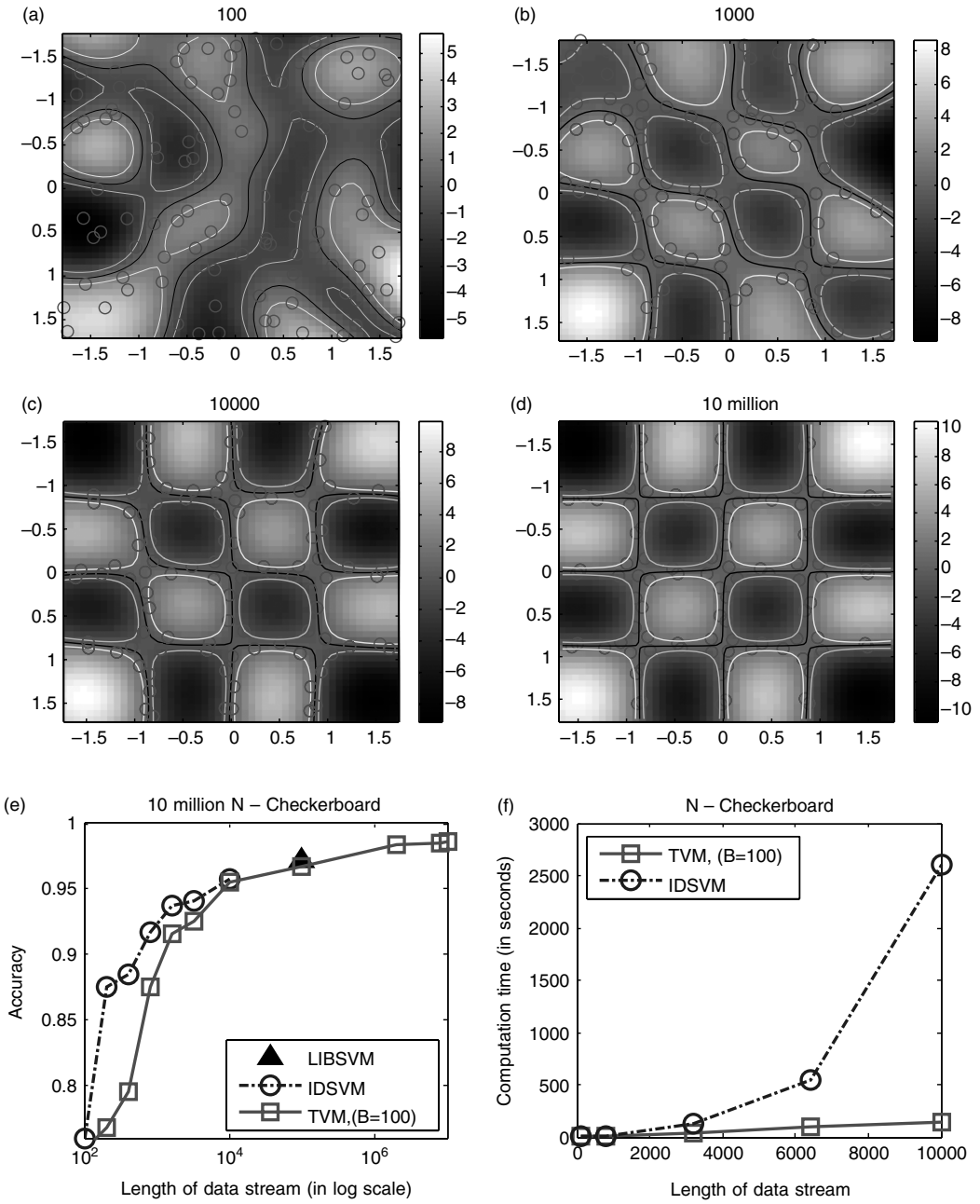

Fig. 3 TVM ($B = 100$) solutions on ten million *N-Checkerboard* data. (a) 100 observed. (b) 1000 observed. (c) 10 000 observed. (d) Ten million observed. (e) Accuracy comparison. (f) Time comparison.

Although different kernels might be appropriate for different datasets, our goal was to compare behavior of different algorithms over a large range of conditions. To keep things simple, and consulting previous SVM evaluations, LIBSVM, IDSVM, and SVM$^{\mathrm{R}}$ used the slack parameter $C = 1$ for *Adult* and $C = 100$ for the remaining 11 datasets. Similarly, following the strategy explained in Section 7, TVM kept C^* at B for *Adult* and at $100B$ for the remaining 11 datasets. RBF kernel width was set to $\delta = M/2$ [24], where M is the number of attributes, for all datasets except the two *Checkerboard* datasets. There, the width was set to 0.37. Polynomial kernel degree d was set to 3 in all datasets except the two *Checkerboard* datasets. There, the degree was set to $d = 7$ because of the complexity of the problem. For the Forgetron, RBF kernel width was optimized using cross-validation for better performance. All experiments were run on a 3G RAM, 3.2 GHz Pentium Dual Core 2 PC.

8.3. TVM Trained on 10 Million *N-Checkerboard* Examples

In Fig. 3(a)–(f), we illustrate the TVM solutions with budget $B = 100$ and RBF kernel after $N = 100$, 1000, 10 000, and 10 million examples were observed from the data stream. Yellow and cyan lines are positive and negative margins, respectively. Black line is the TVM boundary, and red circles are twin prototypes. It can be seen that the TVM decision boundary dramatically improves during the process. This success can be attributed to the improved positioning of twin prototypes near the decision boundary. It is interesting to observe that the TVM margin gradually decreased during the process. The solution for $N = 100$ [Fig. 3(a)] corresponds to *Random* and it is evident that TVM managed to substantially improve its accuracy by careful choice of twin prototypes.

In Fig. 3(e), we compare accuracies of IDSVM, LIBSVM, and TVM with $B = 100$ as a function of the stream size. IDSVM was suspended after $N = 10000$ examples and LIBSVM was trained on $N = 100000$ examples because the resource limits of our PC were reached at these values. As seen, TVM had similar accuracy to IDSVM at $N < 10\,000$ and to LIBSVM at $N = 100\,000$, and its accuracy steadily increased to the impressive 98.7% at $N = 10^7$. Finally, in Fig. 3(f), we compare the runtime of TVM with the runtime of IDSVM algorithm at $N < 10\,000$. It could be observed that the TVM runtime appears linear, whereas the IDSVM runtime appears cubic, as expected.

8.4. TVM$^{\mathrm{H}}$ Versus TVM$^{\mathrm{R}}$

The accuracy and training time comparison between TVM$^{\mathrm{H}}$ and TVM$^{\mathrm{R}}$ with $B = 100$ on 12 datasets is presented in Table 2. Each result listed is an average of five repeated experiments. From the results, we can observe that TVM$^{\mathrm{H}}$ and TVM$^{\mathrm{R}}$ achieve very similar accuracy on all the datasets and that the running time of TVM$^{\mathrm{H}}$ is around two times faster than that of TVM$^{\mathrm{R}}$ on most datasets. The results are consistent with the discussion in Section 4.2 and justify the use of TVM$^{\mathrm{H}}$ as the default TVM algorithm.

8.5. Results on Benchmark Datasets

In this section, we summarize performance results on all 12 benchmark datasets. Each result listed in Tables 3, 4, and 5 is an average of five repeated experiments. Table 3 shows the number of support vector/twin prototypes in the final TVM, IDSVM, LIBSVM, and SVM$^{\mathrm{R}}$ classifiers. To speed-up the experiments, we decided to terminate IDSVM after 3000 s. In this case, in the third column we report the number of support vectors and how many training examples were processed by the time of stopping. We can observe that both LIBSVM and IDSVM classifiers had comparable number of support vectors. SVM$^{\mathrm{R}}$ provided much lighter classifiers than LIBSVM and IDSVM on several noisy datasets, which shows the robustness of ramp loss SVM on noisy data. Comparing LIBSVM, IDSVM, and SVM$^{\mathrm{R}}$ with TVM, it is easy to see that in most cases the number of support vectors is much larger than 100, which was the budget of TVM.

In Table 4, we compare accuracies of the competing algorithms using RBF kernels, whereas in Fig. 4 we illustrate how the accuracy of TVM and IDSVM changed with the number of observed training examples. As expected, the memory-unbounded LIBSVM, IDSVM, and SVM$^{\mathrm{R}}$ had the highest accuracy. LIBSVM and SVM$^{\mathrm{R}}$ were slightly more accurate than IDSVM, which could probably be attributed to occasional early stopping of IDSVM. As could be expected, the only set on which SVM$^{\mathrm{R}}$ was significantly better than LIBSVM was *N-Checkerboard*. Importantly,

Table 2. TVM$^{\mathrm{H}}$ versus TVM$^{\mathrm{R}}$ ($B = 100$).

	Accuracy (%)		Training time (s)	
Datasets	TVM$^{\mathrm{H}}$	TVM$^{\mathrm{R}}$	TVM$^{\mathrm{H}}$	TVM$^{\mathrm{R}}$
Adult	82.1	82.3	230	340
Banana	89.8	89.8	21	34
Checkerboard	98.1	98.1	1406	3283
N-Checkerboard	97.1	97.3	1778	3944
Covertype	76.5	76.8	1709	3642
Gauss	81.1	81.2	774	866
IJCNN	97.0	96.8	122	268
Letter	87.6	87.1	138	312
Pendigits	99.1	99.1	33	39
Shuttle	99.8	99.8	11	20
USPS	92.1	92.6	186	357
Waveform	87.7	87.5	304	641

Table 3. The summary of the number of SVs/TVs with RBF kernel.

	The number of SVs/TVs			
Datasets	TVM	IDSVM	LIBSVM	SVM$^{\mathrm{R}}$
Adult	100	3 223$^{(7\mathrm{K})}$	8 578	5 582
Banana	100	1 015	960	391
Checkerboard	100	3 410$^{(8\mathrm{K})}$	4 073	4 073
N-Checkerboard	100	5 440$^{(10\mathrm{K})}$	51 230	4 536
Covertype	100	2 617$^{(5\mathrm{K})}$	38 790	20 911
Gauss	100	3 212$^{(5\mathrm{K})}$	39 650	4 654
IJCNN	100	1 257$^{(23\mathrm{K})}$	2 103	1 850
Letter	100	1 261$^{(7\mathrm{K})}$	2 075	2 043
Pendigits	100	250	250	250
Shuttle	100	384	189	189
USPS	100	1 081	1 077	1 077
Waveform	100	2 497$^{(10\mathrm{K})}$	24 905	22 701

Superscript values indicate the number of examples learned by IDSVM before early stopped after 3000 s.

Table 4. Accuracy (mean and standard deviation) comparison on 12 large datasets by RBF kernel.

	RBF kernel									
	Offline		Online	Budgeted online		TVM with different budgets				
	LIBSVM	SVM^R	IDSVM	Forgetron	*Random*					
Datasets	$B = \infty$	$B = \infty$	$B = \infty$	$B = 100$	$B = 100$	$B = 20$	$B = 50$	$B = 100$	$B = 200$	$B = 500$
Adult	84.3 ± 0.0	84.3 ± 0.0	84.0 ± 0.0	70.7 ± 5.5	79.5 ± 1.0	83.0 ± 0.5	82.9 ± 0.3	82.1 ± 0.4	82.8 ± 0.5	83.7 ± 0.3
Banana	90.2 ± 0.9	89.9 ± 0.9	91.6 ± 0.0	83.7 ± 4.0	86.9 ± 0.9	87.5 ± 1.7	88.3 ± 1.3	89.8 ± 0.9	89.8 ± 1.2	89.9 ± 0.9
Checkerboard	99.8 ± 0.1	99.7 ± 0.1	$99.7^{(8K)} \pm 0.1$	82.4 ± 4.6	82.9 ± 3.6	75.0 ± 2.5	91.3 ± 2.9	98.1 ± 1.2	98.4 ± 0.6	99.0 ± 0.4
N-Checkerboard	96.9 ± 0.5	99.7 ± 0.1	$95.8^{(10K)} \pm 0.2$	70.3 ± 8.2	72.9 ± 8.0	73.3 ± 4.8	91.4 ± 1.7	97.1 ± 1.1	97.7 ± 2.1	98.8 ± 1.5
Covertype	85.3 ± 0.0	85.5 ± 0.0	$80.5^{(5K)} \pm 0.1$	56.4 ± 4.9	64.4 ± 2.3	66.9 ± 1.0	71.4 ± 0.5	76.5 ± 0.3	77.9 ± 0.1	79.6 ± 0.1
Gauss	81.5 ± 1.4	81.0 ± 1.4	$80.7^{(5K)} \pm 0.2$	77.3 ± 3.7	78.7 ± 1.2	81.3 ± 0.8	81.3 ± 1.5	81.1 ± 0.8	81.4 ± 0.4	81.8 ± 0.4
IJCNN	98.3 ± 0.1	98.4 ± 0.1	$98.4^{(23K)} \pm 0.0$	88.1 ± 7.2	90.3 ± 1.7	93.1 ± 0.6	95.8 ± 0.7	97.0 ± 0.3	97.4 ± 0.2	97.9 ± 0.1
Letter	97.0 ± 0.4	96.9 ± 0.3	$95.0^{(7K)} \pm 0.2$	73.1 ± 2.4	71.9 ± 2.4	76.5 ± 1.8	82.5 ± 1.5	87.6 ± 1.0	90.3 ± 0.9	92.8 ± 0.2
Pendigits	99.7 ± 0.0	99.7 ± 0.0	98.7 ± 0.0	96.6 ± 2.0	93.9 ± 1.6	96.6 ± 2.5	98.7 ± 0.5	99.1 ± 1.0	99.1 ± 0.1	99.2 ± 0.2
Shuttle	99.9 ± 0.0	99.9 ± 0.0	99.9 ± 0.0	99.4 ± 0.6	98.3 ± 0.5	99.7 ± 0.1	99.6 ± 0.3	99.8 ± 0.0	99.8 ± 0.0	99.8 ± 0.0
USPS	97.3 ± 0.0	97.3 ± 0.0	96.7 ± 0.0	83.9 ± 4.0	86.8 ± 1.6	85.0 ± 2.1	88.5 ± 1.7	92.1 ± 0.5	93.5 ± 0.5	95.0 ± 0.3
Waveform	87.9 ± 0.5	87.8 ± 0.3	$89.0^{(10K)} \pm 0.2$	82.5 ± 1.8	85.4 ± 1.6	86.9 ± 2.4	86.1 ± 0.6	87.7 ± 0.5	88.5 ± 0.4	88.8 ± 0.3
Average	93.2	93.3	92.5	80.4	82.7	83.7	88.2	90.7	91.4	92.2

Superscript values indicate the number of training examples learned by IDSVM after 3000 s.

Table 5. Accuracy comparison on 12 large datasets by polynomial and linear kernels.

	Polynomial				Linear			
	IDSVM	Forgetron	*Random*	TVM	IDSVM	Forgetron	*Random*	TVM
Datasets	$B = \infty$	$B = 100$	$B = 100$	$B = 100$	$B = \infty$	$B = 100$	$B = 100$	$B = 100$
Adult	$72.4^{(7K)}$	71.8	78.1	81.2	$84.5^{(9K)}$	61.9	74.4	81.3
Banana	76.3	61.6	76.2	83.0	54.2	48.8	55.1	56.4
Checkerboard	$99.8^{(70K)}$	54.1	81.9	91.1	$49.7^{(5K)}$	49.8	48.7	51.7
N-Checkerboard	$69.9^{(8K)}$	49.2	71.6	93.9	$49.2^{(3K)}$	50.0	48.8	50.8
Covertype	$71.6^{(6K)}$	60.0	59.1	73.9	$75.7^{(10K)}$	59.4	66.9	75.6
Gauss	$80.9^{(15K)}$	71.4	78.6	81.1	$76.2^{(15K)}$	65.7	75.4	76.9
IJCNN	$95.1^{(24K)}$	84.1	88.0	96.9	92.1	52.5	89.8	94.2
Letter	88.6	68.5	68.0	84.7	$72.2^{(11K)}$	63.9	66.8	72.5
Pendigits	98.3	95.0	93.2	98.8	90.7	82.6	85.9	90.1
Shuttle	99.9	93.7	95.9	99.6	98.1	86.6	95.9	98.1
USPS	95.8	83.3	86.6	92.8	$86.6^{(7K)}$	75.4	78.7	83.8
Waveform	$81.6^{(5K)}$	75.9	78.3	86.6	$85.2^{(17K)}$	65.1	79.6	85.3
Average	85.9	72.4	79.6	88.6	76.2	63.5	72.2	76.4

Superscript values in IDSVM column indicate the number of training examples learned by IDSVM after 3000 s.

TVM accuracies were very competitive and, in most cases, comparable with the memory-unbounded SVMs. TVM accuracies consistently increased with budget B. It is worth noting that TVM achieved quite impressive accuracies for a very modest budget of $B = 100$, whereas TVM with still modest budget of $B = 500$ closely matched the accuracies of the memory-unlimited competitors. Compared with the baseline *Random* algorithm, TVM with $B = 100$ was considerably more accurate. In fact, even TVM with $B = 20$ was superior to *Random*. Interestingly, the perceptron-based algorithm Forgetron with $B = 100$ was not competitive. The standard deviation in accuracy was the highest for the most inaccurate algorithms. As expected, IDSVM's standard deviation was close to zero because it achieves the optimal solution regardless of the ordering of stream data.

Let us briefly discuss some specific results from Table 4. TVM was more accurate than LIBSVM on *N-Checkerboard* data, which confirms its relationship with the SVM^{R}. Similarly, impressive behavior was observed on other two noisy datasets, *Gauss* and *Waveform*. In this case, TVM with a median budget was even successful than SVM^{R} and LIBSVM that both created thousands of support vectors. At the other end of the spectrum are *Letter, USPS*, and *Covertype* data that represent highly complex concepts and have a low to modest level of noise. On *Letter* and *USPS*, accuracy of TVM consistently and significantly increased with the budget size and with $B = 500$ it came close to that of LIBSVM, SVM^{R}, and IDSVM. On *Covertype*, the difference was large even with $B = 500$, which is understandable considering that about 20% of the 100 000 training examples were used as support vectors in SVM^{R} and about 40% in LIBSVM.

A glance at Fig. 4 shows further details specific for each of the benchmark datasets. Figures show another strength of TVM—for every choice of budget B and every dataset, the accuracy of TVM consistently grew with the data stream size and was significantly larger than *Random* (represented by the initial point of each TVM curve). Table 5 compares accuracies of four competing algorithms with polynomial and linear kernels. It is evident that the accuracies were consistently and substantially lower than when RBF kernel was used. Moreover, some results for the polynomial kernels appeared quite erratic. This clearly indicates that RBF kernel is the most suitable choice for the 12 datasets we examined. Despite this, the relative differences between the four algorithms were consistent with the results in Table 4. Namely, TVM accuracy with $B = 100$ was quite comparable with IDSVM, and it was much higher than accuracies of Forgetron and *Random*. This result indicates that the impressive TVM performance is not a function of the specific kernel choice.

8.6. Control of C

Figure 5 illustrates the performance of TVM with and without the control of C method on *Gauss* and *N-Checkerboard* datasets. From Fig. 5(a) and (b), we can observe that with the default value $C^* = 100B$ the accuracy of TVM that controls C nicely grows with the data stream length, whereas fixing $C = 100$ causes TVM accuracy to grow only up to a certain point, after which it sharply degrades. To gain a further insight, we compared choices $C = 10$ and $C^* = 10B$ [Fig. 5(c)] and $C = 1000$ and $C^* = 1000B$ [Fig. 5(d)] on *N-Checkerboard* data. In both cases, TVM with control of C behaved well—using $C^* = 1000B$ resulted in similar accuracy to the default $C^* = 1000B$ choice, whereas the accuracy with $C^* = 1000B$ was somewhat lower. On the other hand, TVM with

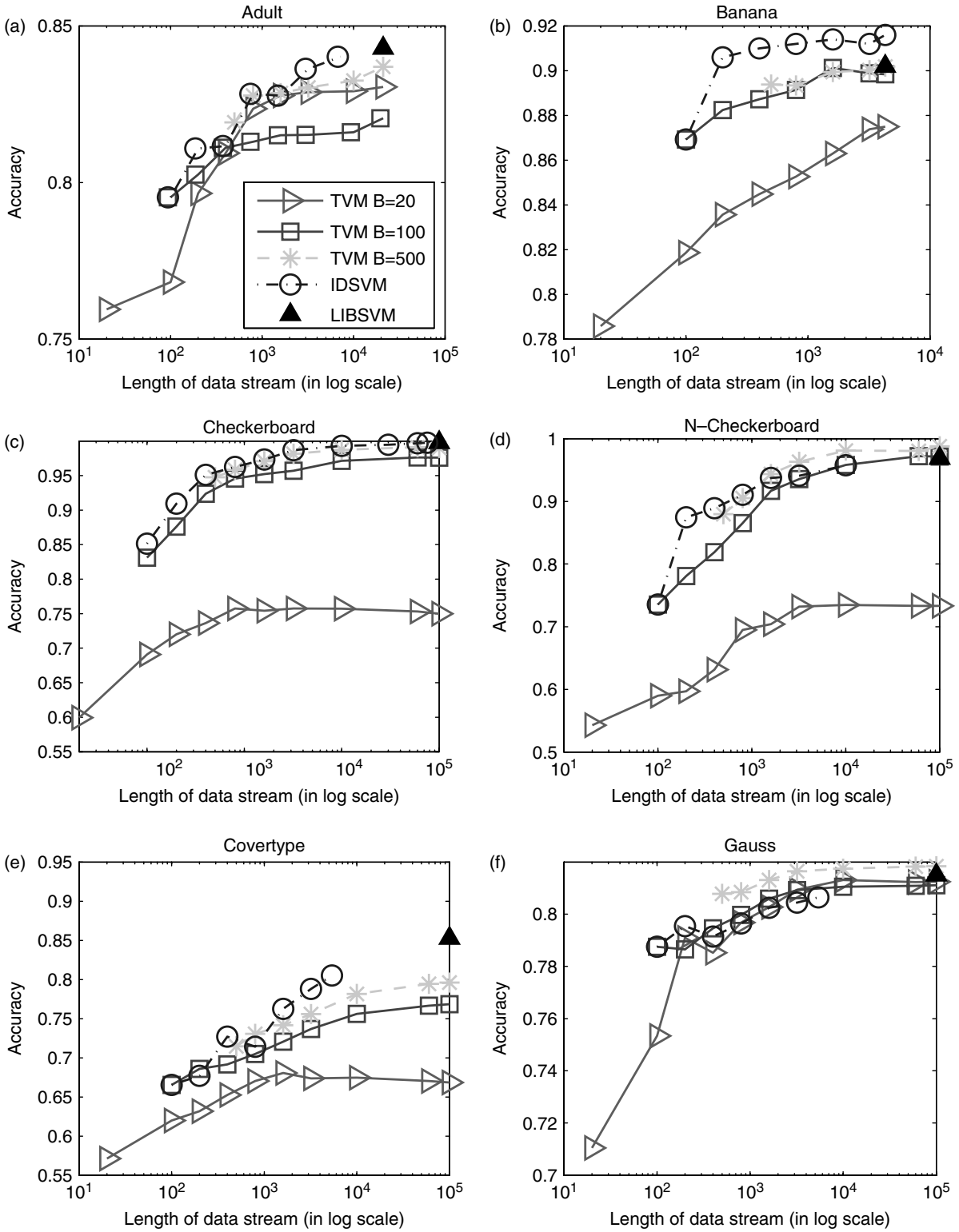

Fig. 4 The accuracy comparison on 12 large datasets (Budget = 20, 100, 500 for TVM). (a) *Adult*. (b) *Banana*. (c) *Checkerboard*. (d) *N-Checkerboard*. (e) *Covertype*. (f) *Gauss*. (g) *IJCNN*. (h) *Letter*. (i) *Pendigits*. (j) *Shuttle*. (k) *USPS*. (l) *Waveform*.

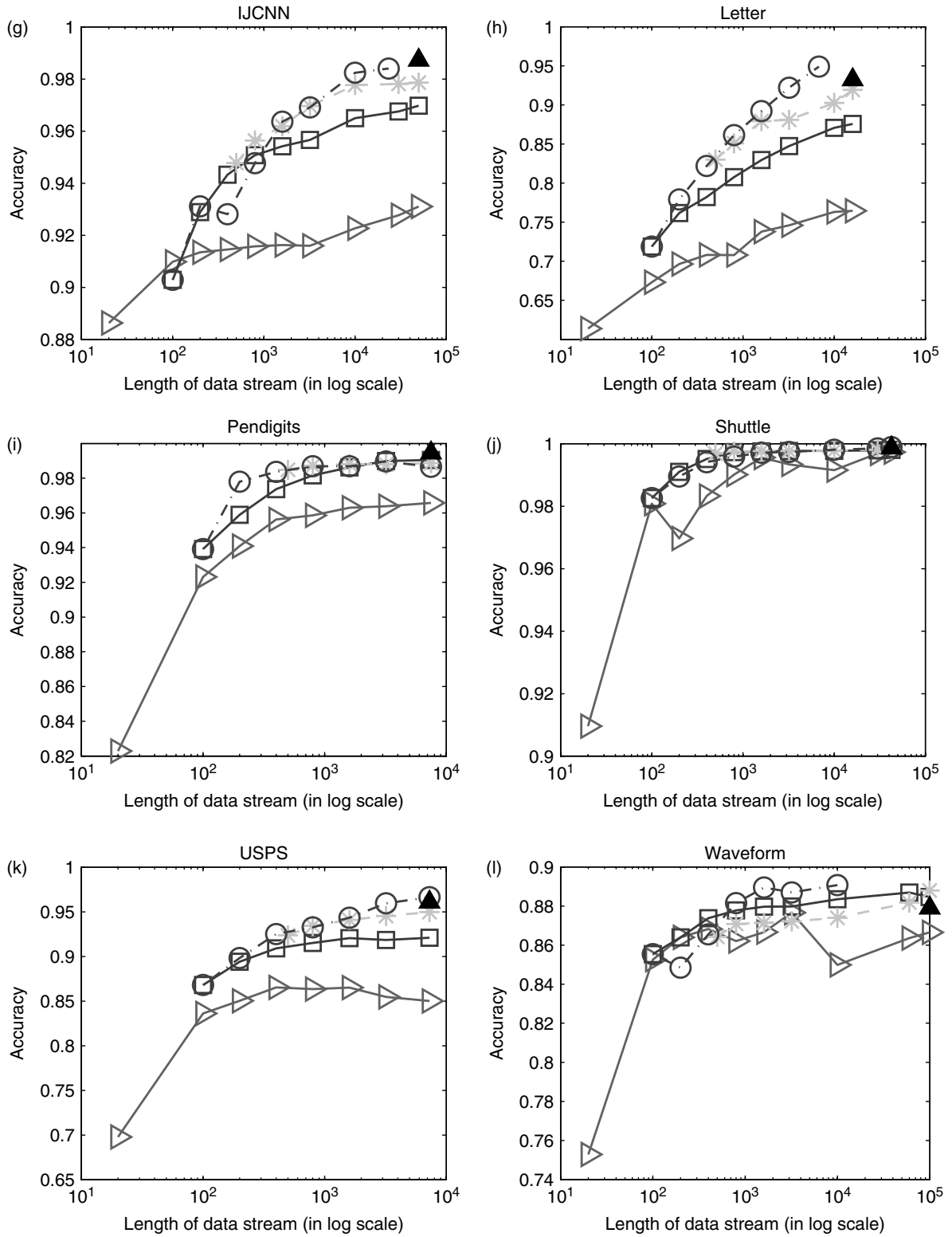

Fig. 4 Continued.

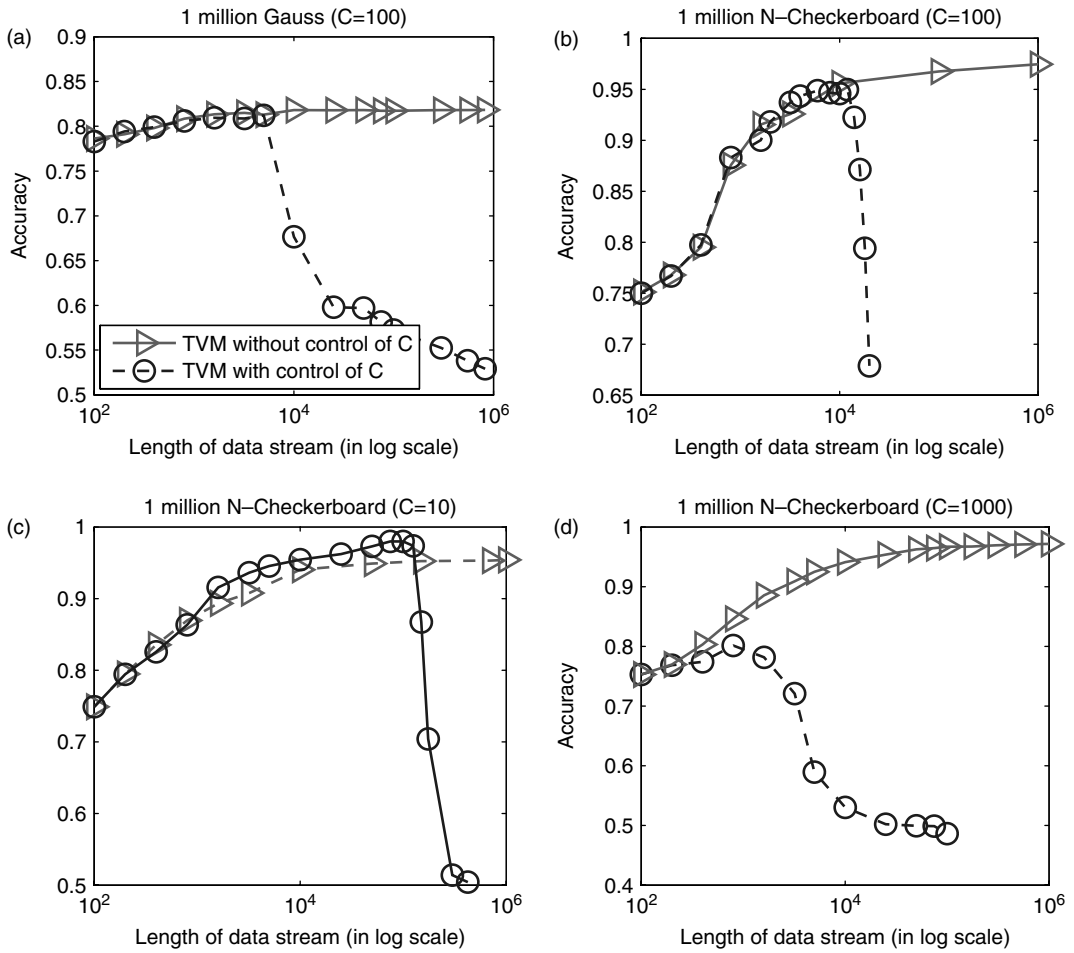

Fig. 5 With control of C versus without control of C. (a) *Gauss* ($C = 100$). (b) *N-Checkerboard* ($C = 100$). (c) *N-Checkerboard* ($C = 10$). (d) *N-Checkerboard* ($C = 1000$).

fixed C again suffered a catastrophic drop in accuracy. Comparing Fig. 5(b)–(d), we can see the pattern in the accuracy drop because it occurred after about 10^3 examples with $C = 1000$, after 10^4 examples with $C = 100$, and after 10^5 examples with $C = 10$. Therefore, it could be expected that the accuracy drop would occur for any fixed choice of C in TVM and that the higher it is the sooner it would happen. This result clearly indicates the importance of controlling the C parameter in TVM algorithm.

8.7. Evaluation of TVM Default Parameters

Table 6 compares performance of the default TVM with TVM where some alternative choices were made. In all cases, the budget was set to $B = 100$ and RBF kernel was used. The first two columns serve as the lower and upper bound on accuracy. The third column is TVM with default values. The alternatives kept all but one of the default choices intact. As could be seen, the default TVM had the highest overall accuracy. Reducing m_2 from 2 to 1 (column 4) removed the buffer zone and it negatively impacted the accuracy. Choice of m_1 appears very important because its increase from 1 to 2 (column 5) resulted in the largest drop in accuracy. This clearly indicates the success of using ramp loss to filter the noisy examples in the design of TVM. Similarly, omission of *One-Sided* merging (column 7) negatively impacted the accuracy. Finally, feature space merging (column 8) resulted in very similar accuracy to input space merging. This outcome justifies the default choice of input space merging, which is computationally cheaper. All these results indicate that the proposed merging strategies are highly successful and that each of them significantly contributes to the success of TVM.

8.8. Evaluation of TVM Prototype Selection and Placement

In Table 7, we give a further insight into budget maintenance procedures of TVM, when $B = 100$ and RBF kernel

Table 6. Accuracy comparison of different TVM versions.

	RBF kernel							
	IDSVM	*Random*	TVM with different heuristics ($B = 100$)					
Datasets	$B = \infty$	$B = 100$	Default	[−1 1]	[−2 2]	No rejection	Global	Feature space
Adult	84.0	79.5	82.1	81.4	80.5	83.2	82.3	81.8
Banana	91.6	86.9	89.8	88.7	89.8	90.5	90.4	90.3
Checkerboard	$99.7^{(8K)}$	82.9	98.1	97.3	95.2	97.9	95.5	97.4
N-Checkerboard	$95.8^{(10K)}$	72.9	97.1	97.1	*89.1*	97.2	94.6	96.9
Covertype	$80.5^{(5K)}$	64.4	76.5	76.6	*72.7*	75.6	*70.3*	76.0
Gauss	$80.7^{(5K)}$	78.7	81.1	81.2	81.2	81.2	81.2	81.2
IJCNN	$98.4^{(23K)}$	90.3	97.0	97.3	*92.1*	*90.0*	95.9	96.8
Letter	$95.0^{(7K)}$	71.9	87.6	87.5	*78.4*	*78.8*	*80.4*	86.9
Pendigits	98.7	93.9	99.1	98.9	*93.0*	98.8	98.6	99.2
Shuttle	99.9	98.3	99.8	99.8	98.0	99.8	99.8	99.8
USPS	96.7	86.8	92.1	91.7	*84.3*	*81.9*	91.1	91.9
Waveform	$89.0^{(10K)}$	85.4	87.7	87.1	*83.2*	*84.3*	88.1	87.6
Average	92.5	82.7	90.7	90.4	86.5	88.3	89.0	90.5

Superscript values indicate the number of training examples learned by IDSVM after 3000 s. Numbers in italics in TVM columns indicate that accuracy is more than 3% lesser than for the default TVM.

are used. The *Labels* column shows the total number of examples being accepted by *Accept* routine of Algorithm 2 (used in line 3 of Algorithm 1). Comparing with the stream length, it can be seen that only a small fraction of examples is accepted by TVM. This indicates that TVM is computationally efficient because examples that are not accepted are simply ignored. This property also indicates that TVM can be useful in active learning [25] where the objective is to maximize accuracy while minimizing the labeling costs. The *Removal* column shows how many times TVM removed an example that was outside the buffer zone (lines 9–11 in Algorithm 1). As expected, removal occurred rarely and during the initial portion of the stream, such as while removing portion of the first B examples that were far from the decision boundary. The *Buffer* column corresponds to number of twin prototypes that remained within the buffer region ($1<|TVM(q)|<2$) at the end of training. We can observe that only a small fraction (between 0% and 24%) of twin prototypes was in the buffer region. This happens because *Accept* only adds new prototypes within the margin region, thus increasing the weight of prototypes within the margin region and keeping weights of prototypes within the buffer region unchanged. As a consequence, prototypes within the buffer region are effectively being attracted to the margin region. The *Rejection* column shows the number of times merging was rejected. As can be seen, rejection happens more frequently in *IJCNN, Letter, USPS*, and *Waveform* than in other datasets. Interestingly, by looking at Table 6 not using *Rejection* on these four datasets resulted in a large decrease in accuracy.

Table 7. Detailed results of TVM ($B = 100$).

Datasets	Labels	Removal	Buffer	Rejection
Adult	5 902	32	1	44
Banana	755	110	11	0
Checkerboard	20 459	111	0	12
N-Checkerboard	23 257	95	0	7
Covertype	30 125	63	0	23
Gauss	11 087	112	23	0
IJCNN	3 711	138	2	545
Letter	4 710	31	3	79
Pendigits	1 650	9	24	0
Shuttle	983	84	17	19
USPS	3 206	0	18	51
Waveform	11 355	60	6	73

9. RELATED WORK

In this section we summarize the related work. We will first discuss online algorithms for two major types of kernel algorithms: quasi-additive and SVM. Then, we summarize the recent work on budget kernel algorithms.

9.1. Online SVM Training

SVM is originally defined as an offline algorithm that has an access to the whole dataset. Recently, two types of approaches were proposed for online training of SVM. In the exact solution approach, the goal is to maintain the optimal SVM solution on all previously seen examples throughout the whole training process. IDSVM [19] and its state-of-the-art implementation [26] are representatives of this approach. The high computational cost of the exact solution approach makes it less practical for large-scale

learning problems. As an alternative, in the approximate solution approach, the requirement for the optimal solution is traded with scalability. To reduce the computational burden, algorithms such as incremental SVM [27] and LASVM [28,29] delete examples that are deemed less likely to become support vectors. However, they do not bound the number of support vectors, and the potential unlimited growth in number of support vectors could still limit its use for learning on high-speed data streams.

9.2. Quasi-Additive Online Algorithms

Perceptron [2] is a classic online learning algorithm which is updated by simply adding the misclassified example to the model weight. Perceptron belongs to a wider class of quasi-additive online algorithms that update model in a greedy manner by using only the newest observed example. Popular modern members of this family of algorithms include ALMA [30], ROMMA [31], NORMA [32], PA [33], and Pegasos [34]. A recent study [35] shows that this type of learning leads to incremental increase of the dual SVM objective and that it can be treated as an approximation to offline SVM. Quasi-additive online algorithms are attractive because they are conceptually simple, computationally efficient, and could become very accurate upon kernelization.

9.3. Keeping the SVM Classifiers Simple

Bounding the space complexity of kernel classifiers is important for many applications where prediction efficiency is of concern. SVM reduced set methods [36,37] start by training a standard SVM on the complete data and then find its sparse approximation by minimizing L_2 distance between the original and the approximated SVM. A limitation of reduced set methods is that it requires training of the full-scale SVM that can be computationally infeasible on large data. Directly training reduced classifier from scratch was proposed in [12,38,39] by reformulating the optimization problem. The basic idea is to train SVM with minimal risk on the complete data under the constraint that the model weight is spanned by only a fixed number of examples. Recently, a similar method to build reduced SVM classifier based on forward selection was proposed in Ref. [40]. This method proceeds in an iterative fashion that greedily selects the example to be added to the model. Preprocessing is another class of approaches for classifier complexity reduction. Preclustering method has been widely used in Refs. [16,32,41]. The motivation is to train SVM on the high-quality summarized data. It is worth mentioning that our proposed prototype summary method in Section 3 also belongs to this family. Recently, another preprocessing method was proposed in Ref. [42], where a simple and fast algorithm (k-nearest-neighbor) is used to pre-select the most informative examples for the following expensive SVM training. Unlike our work, all these reduction methods are offline because they need the access to all the training data.

9.4. Online Learning on a Budget

The pioneering work in budgeted learning for online kernel algorithms addressed the kernel perceptron [5]. To maintain the budget, the support vector[4], which would be predicted most confidently, is removed when the budget is exceeded. The related strategies include removing a random SV [4,7], the oldest SV [6], and the SV that would results in the smallest prediction error [8,43]. Projection is another popular budget maintenance strategy and has been widely adopted by many online kernel algorithms [44–46]. The idea is to minimize the model weight degradation caused by removal of one SV through updating the coefficient of the remaining ones. Instead of enforcing a fixed budget, these algorithms usually allow growing classifier complexity according to the projection quality during training. The problem of budgeted learning with SVM has been addressed only sparingly. In a recent paper [47], a budgeted online SVM algorithm was proposed based on IDSVM. The budget maintenance is achieved by removing the SV farthest away from its nearest neighbor.

10. CONCLUSION AND FUTURE WORK

In this paper, we presented a novel SVM algorithm called TVM for online learning on a budget. TVM achieves sublinear training time and constant space scaling with the data stream size. Experimental results showed that TVM achieves highly competitive accuracy as compared with the memory-unbounded SVM algorithms. This hints at the possibility of building accurate SVM classifiers from arbitrarily large data streams while operating under a very limited memory budget. Furthermore, TVM results in very compact SVM predictors and it directly addresses a problem often observed in practice where the size of SVM grows with the training data size. There are several questions for future research. (i) The tuning of C parameter for online SVM training is still an open question in the SVM community. In an offline learning scenario, a user has a luxury of observing the whole dataset and performing cross-validation to determine C. Our current method addresses the problem of accuracy drop by controlling slack parameter C through fixing C^* (the product of C and all prototype weights). The

[4] In the quasi-additive framework, SVs are all examples added to the model.

remaining open problem is how to determine C^* during the online learning on a budget, when very little might be known *a priori* about the data. (ii) TVM was evaluated on data streams that were sampled randomly from the underlying distribution. The open question is how robust TVM is when this assumption is broken. It would be interesting to explore scenarios with nonrandom sampling and scenarios with changing concepts. (iii) Another interesting question is related to the quadratic memory scaling of TVM with budget size. The space complexity of quasi-additive online algorithms scales only linearly with the budget. However, as seen, performance of the state-of-the-art Forgetron algorithm was not impressive. We are currently exploring if some of the ideas used in TVM could improve performance of budgeted quasi-additive online algorithms.

Acknowledgment

We thank Shai Shalev-Shwartz for sharing the Forgetron code. This work was supported by the US National Science Foundation Grant IIS-0546155.

REFERENCES

[1] J. S. Viter, Random sampling with a reservoir, ACM Trans Math Software 11 (1985), 37–57.
[2] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, Psychol Rev 65 (1958), 386–408.
[3] S. Haykin, Adaptive Filter Theory, New Jersey, Prentice Hall, 2002.
[4] N. Cesa-Bianchi and C. Gentile, Tracking the best hyperplane with a simple budget perceptron, Mach Learn 69 (2007), 143–167.
[5] K. Crammer, J. Kandola, and Y. Singer, Online classification on a budget, In Advances in Neural Information Processing Systems, Vol. 16, Cambridge, MA, MIT Press, 2004, 225–232.
[6] O. Dekel, S. S. Shwartz, and Y. Singer, The forgetron: a kernel-based perceptron on a budget, SIAM J Comput 37 (2008), 1342–1372.
[7] S. Vucetic, V. Coric, and Z. Wang, Compressed kernel perceptrons, In Proceedings of Data Compression Conference, 2009, 153–162.
[8] Z. Wang and S. Vucetic, Tighter perceptron with improved dual use of cached data for model representation and validation, In Proceedings of International Joint Conference on Neural Networks, 2009, 2766–2771.
[9] V. N. Vapnik, Statistical Learning Theory, New York, John Wiley and Sons, Inc., 1998.
[10] J. L. Balcázar, Y. Dai, J. Tanaka, and O. Watanabe, Provably fast training algorithms for support vector machines, Theor Comput Syst 42 (2008), 568–595.
[11] R. Collobert, F. Sinz, J. Weston, and L. Bottou, Trading convexity for scalability, In Proceedings of International Conference on Machine Learning, 2006, 201–208.
[12] Y.-J. Lee and O. L. Mangasarian, RSVM: reduced support vector machines, In Proceedings of SIAM Conference on Data Mining, 2001.
[13] D. Pavlov, D. Chudova, and P. Smyth, Towards scalable support vector machines using squashing, In Proceedings of ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2000, 295–299.
[14] I. W. Tsang, J. T. Kwok, and P.-M. Cheung, Core vector machines: fast SVM training on very large data sets, J Mach Learn Res 8 (2005), 291–301.
[15] S. V. N. Vishwanathan, A. J. Smola, and M. N. Murty. SimpleSVM, In Proceedings of International Conference on Machine Learning, 2003, 760–767.
[16] H. Yu, J. Yang, J. Han, and X.-L. Li, Making SVMs scalable to large data sets using hierarchical cluster indexing, Data Mining Knowl Discov 11 (2005), 295–321.
[17] I. Steinwart, Sparseness of support vector machines, J Mach Learn Res 4 (2003), 1071–1105.
[18] A. L. Yuille and A. Rangarajan, The concave convex procedure (CCCP), In Advances in Neural Information Processing Systems, Vol. 14, Cambridge, MA, MIT Press, 2002, 409–415.
[19] G. Cauwenberghs and T. Poggio, Incremental and decremental support vector machine learning, In Advances in Neural Information Processing Systems, Vol. 13, Cambridge, MA, MIT Press, 2001, 409–415.
[20] Z. Wang and S. Vucetic, Fast online training of ramp loss support vector machines, In Proceedings of International Conference on Data Mining, 2009, 569–577.
[21] D. Nguyen and T. Ho, An efficient method for simplifying support vector machines, In Proceedings of International Conference on Machine Learning, 2005, 617–624.
[22] T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu, The entire regularization path for the support vector machine, J Mach Learn Res 5 (2004), 1391–1415.
[23] C. P. Diehl and G. Cauwenberghs, SVM incremental learning, adaptation and optimization, In Proceedings of the International Joint Conference on Neural Networks, 2003, 2685–2690.
[24] C.-C. Chang and C.-J. Lin, LIBSVM : a library for support vector machines, http://www.csie.ntu.edu.tw/∼cjlin/libsvm, [Last accessed 2001].
[25] S. Tong and D. Koller, Support vector machine active learning with applications to text classification, J Mach Learn Res 2 (2001), 45–66.
[26] P. Laskov, H. Gehl, S. Krüger, and K.-R. Müller, Incremental support vector learning: analysis, implementation and applications, J Mach Learn Res 7 (2006), 1909–1936.
[27] C. Domeniconi and D. Gunopulos, Incremental support vector machine construction, In Proceedings of International Conference on Data Mining, 2001, 589–592.
[28] A. Bordes, S. Ertekin, J. Wesdon, and L. Bottou, Fast kernel classifiers for online and active learning, J Mach Learn Res 6 (2005), 1579–1619.
[29] G. Loosli, S. Canu, and L. Bottou, Training invariant support vector machines using selective sampling, In Large Scale Kernel Machines, Cambridge, MA, MIT Press, 2007, 301–320.
[30] C. Gentile, A new approximate maximal margin classification algorithm, J Mach Learn Res 2 (2001), 213–242.
[31] Y. Li and P. Long, The relaxed online maximum margin algorithm, Mach Learn 46 (2002), 361–387.
[32] J. Kivinen, A. J. Smola, and R. C. Williamson, Online learning with kernels, IEEE Trans Signal Process 52 (2002), 2165–2176.
[33] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, Online passive-agressive algorithms, J Mach Learn Res 7 (2006), 551–585.

[34] S. Shalev-Shwartz, Y. Singer, and N. Srebro, Pegasos: primal estimated sub-gradient solver for svm, Proceedings of International Conference on Machine Learning, 2007 807–814.

[35] S. Shalev-Shwartz and Y. Singer, Online learning meets optimization in the dual, In Proceedings of Annual Conference on Learning Theory, 2006, 423–437.

[36] C. J. C. Burges, Simplified support vector decision rules, In Proceedings of International Conference on Machine Learning, 1996, 71–77.

[37] B. Schökopf, S. Mika, C. J. C. Burges, P. Knirsch, K. Müler, G. Räsch, and A. J. Smola, Input space versus feature space in kernel-based methods, IEEE Trans Neural Networks 10 (1999), 1000–1017.

[38] O. Dekel and Y. Singer, Support vector machines on a budget, In Advances in Neural Information Processing Systems, Vol. 19, Cambridge, MA, MIT Press, 2006, 345–352.

[39] M.-R. Wu, B. Schökopf, and G. Barik, Building sparse large margin classifiers, In Proceedings of International Conference on Machine Learning, 2005, 996–1003.

[40] S. S. Keerthi, O. Chapelle, and D. DeCoste, Building support vector machines with reduced classifier complexity, J Mach Learn Res 7 (2006), 1493–1515.

[41] B. Li, M. Chi, J. Fan, and X. Xue, Support cluster machine, In Proceedings of International Conference on Machine Learning, 2007, 505–512.

[42] H. Shin and S. Cho, Neighborhood property based pattern selection for support vector machines, Neural Comput 19 (2007), 816–855.

[43] J. Weston, A. Bordes, and L. Bottou, Online (and offline) on an even tighter budget, In Proceedings of International Workshop on Artificial Intelligence and Statistics, 2005, 413–420.

[44] Y. Engel, S. Mannor, and R. Meir, Sparse online greedy support vector regression, In Proceedings of European Conference on Machine Learning, 2002, 84–96.

[45] F. Orabona, J. Keshet, and B. Caputo, The projectron: a bounded kernel-based perceptron, In Proceedings of International Conference on Machine Learning, 2008, 720–727.

[46] H.-Q. Wang, P. Li, F.-R. Gao, Z.-H. Song, and S. X. Ding, Kernel classifier with adaptive structure and fixed memory for process diagnosis, AIChE Journal 52 (2006), 3515–3531.

[47] S. Agarwal, V. V. Saradhi, and H. Karnick, Kernel-based online machine learning and support vector reduction, Neurocomputing 71 (2008), 1230–1237.