Twin Vector Machines for Online Learning on a Budget *

Zhuang Wang   Slobodan Vucetic[†]

Abstract

This paper proposes Twin Vector Machine (TVM), a constant space and sublinear time Support Vector Machine (SVM) algorithm for online learning. TVM achieves its favorable scaling by maintaining only a fixed number of examples, called the twin vectors, and their associated information in memory during training. In addition, TVM guarantees that Kuhn-Tucker conditions are satisfied on all twin vectors at any time. To maximize the accuracy of TVM, twin vectors are adjusted during the training phase 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 twin vector if it is near the decision boundary. If this happens, two twin vectors are selected and merged into a single twin vector to maintain the budget. Finally, TVM is updated by incremental and decremental learning to account for the change. Several methods for twin vector merging were proposed and experimentally evaluated. TVMs were thoroughly tested on 12 large data sets. In most cases, the accuracy of low-budget TVMs was comparable to the state of the art 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 to Forgetron, a popular kernel perceptron algorithm on a budget. The results illustrate that highly accurate online SVMs could be trained from large data streams using devices with severely limited memory budgets.

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 sub-linearly with sample size and dimension, both in runtime and in space. Interestingly, 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, data sets or data streams using memory limited resources.

In this paper, we address the online learning on a budget scenario with the following characteristics: (1) independent and identically distributed training examples are observed sequentially and in a single pass; (2) the learner can choose to maintain in the limited memory any information about the observed examples; (3) 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 maintain in the memory. 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 [16] can be used to maintain a random sample from a data stream. Then, model trained can be done off-line 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 the memory and updated as new examples are observed. An example of the data-free approach is the perceptron algorithm [10] 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 are hybrid in that they require maintenance of both data and model in the memory.

An example of the hybrid approach for linear regression is the recursive least squares algorithm [8] 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. A representative of hybrid approaches in classification are budget kernel perceptrons [2, 4, 5]. The kernel perceptrons are represented by a subset of observed examples (i.e. support vectors) and their weights, and the budget solution is achieved by ensuring that the number of support vectors is bounded.

[*] This work was supported by the U.S. National Science Foundation Grant IIS-0546155.

[†] Center for Information Science and Technology, Temple University, Philadelphia, PA 19122, USA, email: zhuang@temple.edu, vucetic@ist.temple.edu

While there are theoretical guarantees for convergence of budget kernel perceptrons for certain noise-free classification problems, their performance is often poor on noisy classification problems.

In this paper, we propose a Support Vector Machine (SVM)[14] 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. Such scaling is often unacceptable for data mining applications. Various solutions have been proposed [13, 15, 17] to improve the time and space efficiency of SVMs and make them applicable to very large data sets. While 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 the training data, which is not acceptable in the online learning scenario.

The proposed algorithm, called Twin 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. Each example kept in the memory, called the twin vector, represents a summary of the training examples in its neighborhood and thus captures the local data distribution. To optimally utilize the budget, twin vectors are positioned near the decision boundary, which is the most informative region of the input space. The TVM and the set of twin vectors are updated after each newly observed training example while keeping the memory budget as follows. A new example is added as a new twin vector if it is near the decision boundary. If this happens, two twin vectors are selected and combined into a single twin vector to maintain the budget. Finally, TVM is updated such that Kuhn-Tucker conditions are satisfied on all twin vectors at any time. To accomplish this, we used the solution proposed in the online memory-unbounded Incremental-Decremental SVM (IDSVM) [1] algorithm. It uses an efficient adiabatic procedure to update the SVM upon inclusion or deletion of a support vector. The resulting TVM algorithm achieves constant space and linear time scaling, and is very appropriate for online learning from large data sets using a limited memory budget.

The paper is organized as follows. In §2, we give an overview of SVMs and the incremental and decremental learning. In §3, we introduce TVM and in §4 we describe how they can be used for online learning on a budget. §5 provides results of the thorough characterization of TVM on a number of large data sets.

2 Preliminaries

2.1 SVM, Dual Formulation, and Kuhn-Tucker Conditions. The SVM classifier is of the form $f(x) = w \cdot \Phi(x) + b$, where Φ is a nonlinear mapping of the attribute space, and is trained from data set $D = \{(x_i, y_i), i = 1...N\}$ by optimizing the primal problem

$$\begin{aligned} &\min \tfrac{1}{2}||w||^2 + C\sum_i \xi_i \\ &\text{s.t. } y_i(w \cdot \Phi(x_i) + b) \geq 1 - \xi_i, \xi_i \geq 0, \forall i \in \{1, ..., N\} \end{aligned}$$

where ξ_i are the non-negative slack variables and C is the non-negative slack parameter. In the dual formulation of the optimization, the primal problem is transformed to

$$\min_{0 \leq \alpha_i \leq C} : W = \frac{1}{2}\sum_{i,j} \alpha_i Q_{ij} \alpha_j - \sum_i \alpha_i + b\sum_i y_i \alpha_i$$

where α_i 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 $K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)$. The resulting SVM classifier can be conveniently represented in the dual form as

$$f(x) = \sum_{i=1}^{N} y_i \alpha_i K(x_i, x) + b$$

Partial derivatives of the objective function W of the dual problem should satisfy the KT conditions

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

and create a partition of the training data D into three categories: the set S of *margin support vectors* that are strictly on the margin ($y_i f(x_i) = 1$), the set E of *error support vectors* that violate the margin ($y_i f(x_i) < 1$), and the set R of *reserve vectors* that are correctly classified and outside the margin ($y_i f(x_i) > 1$). Therefore, the SVM model is represented by a set of Lagrange multipliers α_i and training examples for which $\alpha_i > 0$ (margin and error support vectors).

2.2 Adiabatic SVM Updates [1]. Given the SVM consisting of N support and reserve vectors and a new example (x_c, y_c), [1] proposes how to change values of $\alpha_i, i = 1, ..., N$, and b such that all KT conditions are maintained. 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 values of α_i's and b should be updated.

By assuming that addition of the new example does not change the 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

$$\begin{aligned} \Delta g_i &= Q_{ic}\alpha_c + \sum_{j \in S} Q_{ij}\Delta\alpha_j + y_i\Delta b \\ 0 &= y_c\alpha_c + \sum_{j \in S} y_j\Delta\alpha_j, \forall i \in D \cup \{c\} \end{aligned} \tag{2.1}$$

Since $\Delta g_i = 0$ for the margin support vector set S with indices $S = \{s_1, ..., s_p\}$, where P is the number of margin support vectors (set S), equations (2.1) could be written in the matrix form as

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

where J is symmetric Jacobian matrix

$$J = \begin{bmatrix} 0 & y_{s_1} & \cdots & y_{s_p} \\ y_{s_1} & Q_{s_1 s_1} & \cdots & Q_{s_1 s_p} \\ \vdots & \vdots & \ddots & \vdots \\ y_{s_p} & Q_{s_p s_1} & \cdots & Q_{s_p s_p} \end{bmatrix}$$

Matrix J is the enlarged (by one row and column) Gram matrix of the 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 adiabatic procedure. The first is finding the optimal value of α_c as the value that minimizes the objective function W. This is accomplished by slowly increasing its value starting from zero as long as W 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 [1]. 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. Decremental unlearning (removing any example from the training set) is implemented through adiabatic reversal of incremental learning.

2.3 Costs of an Adiabatic Update. During each incremental or decremental step, the major computational cost is in calculating the inverse of Jacobian matrix J. Instead of directly inverting J that would take $\mathrm{O}(P^3)$ time, a recursive procedure is used that utilizes the previously calculated inverse of J, in which J differed by a single row and column. The cost of the inversion in such case scales as $\mathrm{O}(P^2)$. Assuming that the amount of bookkeeping at each incremental/decremental step is upper bounded (reasonable assumption in practice), the runtime of incremental learning/decremental unlearning of a single example scales as $\mathrm{O}(P^2)$.

For the memory requirement, an adiabatic update requires to keep the $(P+1) \times (P+1)$ inverse of the Jacobian matrix J, the $N \times (K+1)$ matrix of support and reserve vectors, where K is the number of attributes (to be able to update J^{-1} when a new example arrives, regardless of the future assignments of vectors to S, E, and R sets), the $N \times P$ Gram matrix (for bookkeeping), the $N \times 1$ vector of g values, and the $N \times 1$ vector of Lagrange multipliers α. Assuming the number of margin support vectors P scales as $\mathrm{O}(N)$, both space and runtime of the adiabatic update scale as $\mathrm{O}(N^2)$.

3 Twin Vector Machine

3.1 Twin Vectors. The idea for TVM comes from vector quantization where each example is represented by its quantized version. We will now illustrate how this idea leads to the constant space and linear time SVM. Let us assume that we are given B prototype attribute vectors $q_1, ..., q_B$ and that each training example is represented by its nearest prototype. For the moment, we will not worry about how the prototypes are selected; we will address this question in §4. Through the prototype representation, the original training data set $D = \{(x_i, y_i), i = 1...N\}$ is transformed to $Q(D) = \{(Q(x_i), y_i), i = 1...N\}$, where Q is the quantization function defined as $Q(x) = \{q_k, k = \arg\min_{j=1:B} ||x - q_j||\}$. We note that the assignment to the nearest prototype results in the minimization of attribute distortion, defined as $E(||X - Q(X)||^2)$. Clearly, as the number of prototypes B grows, it is expected that the attribute distortion decreases.

Training SVM on N quantized examples from $Q(D)$ reduces to solving the following primal problem,

$$\begin{aligned} &\min \tfrac{1}{2}||w||^2 + C\sum_j (s_j^+\xi_j^+ + s_j^-\xi_j^-) \\ &\quad w\Phi(q_j) + b \geq 1 - \xi_j^+, \\ &\text{s.t. } -(w\Phi(q_j) + b) \geq 1 - \xi_j^-, \\ &\quad \xi_j^+, \xi_j^- \geq 0, j = \{1, ..., B\}, \end{aligned}$$

where s_j^+ and s_j^- are the numbers of positive and negative examples quantized to prototype q_j, and ξ_j^+ and ξ_j^- are the corresponding slack variables. Therefore, prototype q_j is 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 Vector*. Training an SVM on N quantized examples from $Q(D)$ is analogous to training an SVM on the set of B twin vectors $TV = \{TV_j, j = 1...B\}$. Thus, from the perspective of SVM training, $Q(D) = TV$.

After transformation to the dual problem the KT conditions become

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

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 [1], 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. We call the SVM trained on TV set of twin vectors the offline *Twin Vector Machine* (TVM). The resulting offline TVM predictor can be expressed as

$$TVM(x) = \sum_j (\alpha_j^+ - \alpha_j^-) K(q_j, x) + b.$$

Let us discuss computational costs of offline TVM training with fixed pivot vectors. Because the $Q(D)$ can be regarded as a data set with $2B$ weighted examples, the solution of the quadratic programming problem would have $\mathrm{O}(B^3)$ runtime and require $\mathrm{O}(B^2)$ memory, which implies constant runtime and constant memory scaling with N. By adding the costs of converting D to $Q(D)$ the runtime becomes $\mathrm{O}(B^3) + \mathrm{O}(N)$ which represents linear scaling with data size N.

4 Twin Vector Machine for Online Learning on a Budget

In §3 we showed that quantization of the original data set D to B twin vectors allows SVM training with linear time and constant space scaling with data size. Throughout the section we assumed that the set of B pivot vectors was known and fixed. A natural question addressed in this section is how to choose the pivot vectors in an online fashion. Our idea is to adaptively change the pivot vectors such that they are positioned near the decision boundary. The details of the resulting online TVM (or TVM, for short) algorithm for online training on a budget are as follows.

4.1 The Algorithm. The outline of the algorithm is shown in Algorithm 1. After initializing the TVM to zero predictor and setting TV to empty, examples from the stream are read sequentially. For each observed example, we determine if it is useful, using the *Beneficial* routine. If the example is useful, it is added to the reservoir and the twin vector set TV is updated to maintain the budget (*UpdateTV*). Then, TVM is updated to account for the changes in TV using the incremental and decremental learning (*UpdateTVM*).

Algorithm 1 TVM

```
Input: Data stream $D = \{(x_i, y_i), i = 1...N\}$, budget $B$, kernel function $K$, slack parameter $C$
Output: TVM with parameters $\alpha_1^+, \alpha_1^-, ..., \alpha_B^+, \alpha_B^-$, $b$
$TVM = 0, TV = \oslash$
for i=1 to $N$ do
  if $Beneficial(x_i)$ then
    $UpdateTV$
    $UpdateTVM$
  end if
end for
```

The basic idea of *Beneficial*, which is consistent with the successful approaches for active learning with SVMs [11, 12], is that examples that are near the decision boundary (i.e., $|TVM(x)|$ is small) are most informative. Ignoring examples that are far beyond the margin is justified at two levels: positively classified examples are reserve vectors that are not likely to become support vectors, while negatively classified examples are error support vectors that are most likely noisy and with a negative influence on the classification accuracy. Algorithm 2 describes the *Beneficial* routine. The first B observed examples are labeled as beneficial by default. Others are labeled as beneficial if the absolute value of their TVM prediction is below threshold m_1. As a default value, we use $m_1 = 1$ that results in acceptance of new examples only if they are within the margin of the current TVM.

Algorithm 2 Beneficial

```
if $size(TV) < B$ or $|TVM(x_i)| \leq m_1$ then
  return 1
else
  return 0
end if
```

UpdateTV (Algorithm 3) updates the twin set TV.

It starts by creating the overflow twin vector TV_{B+1} from the observed example (x_i, y_i), where $q_{B+1} = x_i$, $s^+_{B+1} = 1$ and $s^-_{B+1} = 0$ if $y_i = +1$, and $s^+_{B+1} = 0$ and $s^-_{B+1} = 1$ if $y_i = -1$. If there is a space in the reservoir, the overflow twin is added to it. If the reservoir is full (TV has B twin vectors), its twin vectors have to be modified. *UpdateTV* recognizes two scenarios. In case when there is a twin vector that is sufficiently beyond the margin (i.e., $|TVM(q_j)| > m_2$), *UpdateTV* removes it to make space for the overflow twin. The threshold m_2 is selected such that $m_2 > m_1$ (its default value is set to 2, see Figure 1.a) in order to create a buffer zone that prevents premature removal of potentially useful twins.

In case when all B twin vectors are within the buffer zone, *UpdateTV* selects two twin vectors TV_i and TV_j and merges them to create a new twin vector TV_{new} that replaces them. Details of how the two twin vectors are selected are given in §4.2.

Algorithm 3 UpdateTV

```
s = size(TV)
$TV_{B+1} = \{(x_i, y_i, 1), (x_i, -y_i, 0)\}$
if $s < B$ then
    $TV_{s+1} = TV_{B+1}$
else if $\max_{i=1:B} |f(q_i)| > m_2$ then
    $k = \arg\max_{i=1:B} |f(q_i)|$
    $TV_k = TV_{B+1}$
else
    find the best pair $TV_i, TV_j (i < j)$ to merge
    $TV_i = \{(q_{new}, +1, s^+_i + s^+_j), (q_{new}, -1, s^-_i + s^-_j)\}$
    $TV_j = TV_{B+1}$
end if
```

The outcome of *UpdateTV* can be addition of a new twin vector, replacement of a twin vector with the overflow twin, or replacement of one or two twin vectors through merging. *UpdateTVM* performs decremental unlearning of the removed twin vector(s) and incremental learning of the added twin vector(s) such that Kuhn-Tucker conditions are satisfied on all twin vectors during the online learning.

4.2 Merging Methods.

4.2 Merging Methods. In this section we discuss several methods for selecting the best twin vectors for merging in *UpdateTV*.

4.2.1 Merging in Input Space. In case when all B twin vectors within the buffer zone, *UpdateTV* selects two twin vectors TV_i and TV_j and merges them to create a new twin vector TV_{new} that replaces them. Given TV_i and TV_j, one approach is to calculate the pivot vector of TV_{new}, q_{new}, to minimize distortion in the input space defined as

$$d = s_i||q_i - q_{new}||^2 + s_j||q_j - q_{new}||^2,$$

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

$$q_{new} = \frac{s_i q_i + s_j q_j}{s_i + s_j},$$

and that the resulting minimal distortion, denoted as d^*_{ij} is

$$d^*_{ij} = \frac{s_i s_j ||q_i - q_j||^2}{s_i + s_j}.$$

The best pair TV_i and TV_j of twin vectors 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 calls to *UpdateTV*, both costs can be reduced to $O(B)$ in expectation. In fact, for each twin vector 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 *UpdateTV*, we calculate distortions between the newly created twin vectors and the unchanged twin vectors and use that to modify the memorized indices and distortions.

4.2.2 Merging in Feature Space. An alternative to merging in input space is merging in feature space. Let us denote Φ as the mapping of the input space to a feature space induced by the kernel function K, such that $K(x, y) = \Phi(x)^T\Phi(y)$. Given TV_i and TV_j, the goal is to find q_{new} that minimizes feature space distortion defined as

$$d = s_i||\Phi(q_i) - \Phi(q_{new})||^2 + s_j||\Phi(q_j) - \Phi(q_{new})||^2.$$

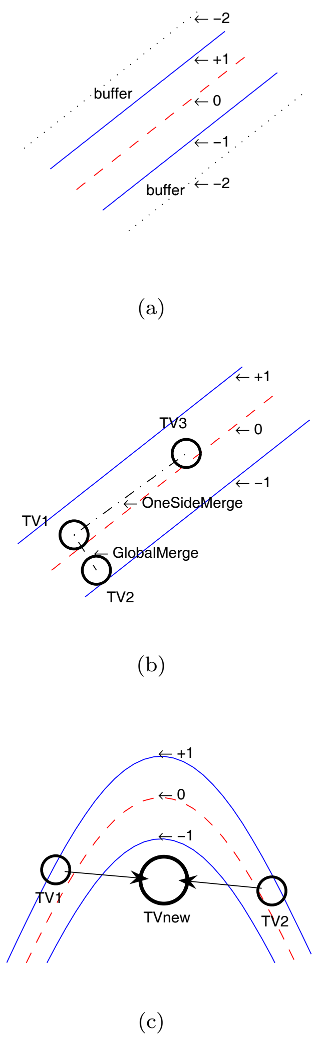
Similarly to results of §4.2.1, the optimal $\Phi(q_{new})$ is

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

The challenge with feature space merging is that the pre-image of $\Phi(q_{new})$ may not exist (as is the case with RBF kernels). As proposed in [9], this issue can be addressed by finding the approximate solution q'_{new} by minimizing

$$\min_{q'_{new}} ||\Phi(q'_{new}) - \frac{s_i\Phi(q_i) + s_j\Phi(q_j)}{s_i + s_j}||^2. \tag{4.2}$$

The general solution of (4.2) is hard to achieve since it requires nonlinear optimization that depends on the choice of kernel. In [9], an algorithm was developed to



(a)

(b)

(c)

Figure 1: Figures of Buffer, Merging and Rejection

solve (4.2) for RBF and polynominal kernels. Interestingly, in both cases, q'_{new} lies on the line connecting q_i and q_j.

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

4.2.3 Global versus One-Sided Merging. Without an additional constraint, TVM allows merging of twin vectors across the decision boundary. Potential issue with such merging is loss of margin support from the original twin vectors and creation of a new twin near the boundary with nearly equal positive and negative weights. This can be harmful to the quality of TVM. In *One-Sided* merging the twin vectors are separated into those in the positive region, $TVM(q_i) > 0$, and the negative region, $TVM(q_i) < 0$. Only pairs of twin vectors from the same region are considered for merging. As an illustration, although merging of TV_1 and TV_2 from Figure 1.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.

4.2.4 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 vectors TV_1 and TV_2 from Figure 1.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 $TVM(q_{new})$ to its estimated value under the linear assumption, $\hat{TVM}(q_{new}) = (s_i TVM(q_i) + s_j TVM(q_j))/(s_i + s_j)$. *Rejection* accepts the merging if $TVM(q_{new})$ is near its estimated value,

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

where $\eta = 0.2$ is used as the default value. If the selected pair of twin vectors fails the test, *Rejection* considers the next best merging candidates. If they pass the test, the merging is executed. Otherwise, *UpdateTV* concludes that merging cannot be successfully performed and the overflow twin is simply ignored.

4.3 Control of C. In the online learning on a budget, an additional challenge is the choice of the slack parameter C, that controls the tradeoff between model complexity and training error. For a fixed C, the influence of training error would grow with data stream size due to increase in weights s_j^+ and s_j^- of twin vectors. Compounded with the bounded number of twin vectors 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 C such that the product $C\sum_j(s_j^+ + s_j^-)$ is kept constant during the online learning process. Adjusting TVM with respect to a change in C could be done efficiently using *path regularization* techniques [7]. In TVM, we use *regularization parameter permutation* [6] which is a method implemented in IDSVM algorithm.

4.4 Costs of TVM Online Training. The memory requirement of TVM is constant in sample size and scales as $O(B^2)$ with the budget B. It consists of three routines which have different runtimes. The *Beneficial* needs to provide TVM prediction for a new example and it has $O(B)$ runtime. In the worst case, *UpdateTV* has to perform merging. Finding the best pair of twin vectors for merging requires $O(B)$ expected runtime subject to appropriate bookkeeping from previous calls of *UpdateTV*. Dynamically adjusting C takes $O(B)$ time. Finally, the runtime of *UpdateTVM* is $O(B^2)$. The total runtime is therefore $O(NB + nB^2)$, where n is the number of examples among the N observed examples which are selected by *Beneficial*. 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 margin decreases with N and *Beneficial* becomes more and more selective. Since nB^2 dominates the runtime and $n = O(N)$, the total runtime appears sublinear in N.

5 Experimental Results

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

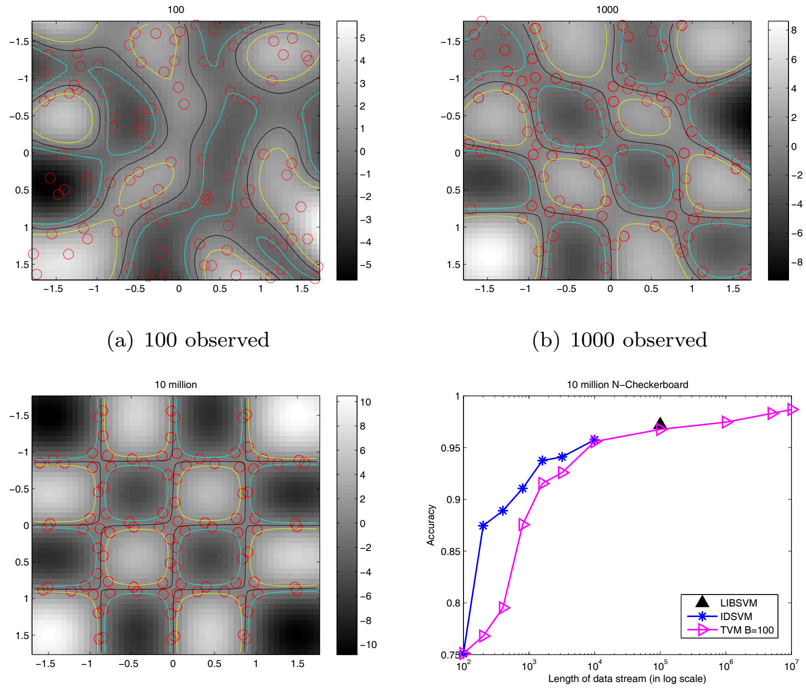
5.1 Data Sets. Properties of 12 benchmark data sets for binary classification are summarized in Table 1. The multi-class data sets were converted to two-class sets as follows. For the digit datasets *Pendigits* and *USPS* we converted the original 10-class problems to binary by representing digits 1, 2, 4, 5, 7 (non-round 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 7-class *Shuttle* data set was converted to binary data by representing class 1 as negative and the remaining ones as negative. Class 1 in the 3-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 6 classes as negative. *Adult*, *Banana*, *Gauss*, and *IJCNN* were originally 2-class data sets. Checkerboard data was generated as a uniformly distributed two-dimensional 4×4 checkerboard with alternating class assignments (see Figure 3). *Checkerboard* is a noise-free data set 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 data sets, 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 data sets were scaled to mean zero and standard deviation 1.

Table 1: Data set summaries

Datasets	Training Size	Test Size	Attributes
Adult	21048	9114	123
Banana	4300	1000	2
Checkerboard	100000	5000	2
N-Checkerboard	100000	5000	2
Covertype	100000	10000	54
Gauss	100000	5000	2
IJCNN	49990	91701	22
Letter	16000	4000	16
Pendigits	7494	3498	16
Shuttle	42603	14167	9
USPS	7291	2007	256
Waveform	100000	5000	21

5.2 Evaluation Procedure. We used three SVM algorithms (LIBSVM [3], IDSVM [1], and TVM) and one kernel perceptron algorithm (Forgetron [5]) in the experiments. Because IDSVM is a memory-unconstrained online algorithm it served as an upper bound on accuracy achievable by TVM. LIBSVM, one of the most successful batch-mode SVM algorithms, is both highly accurate and computationally efficient. Although it is not an online algorithm, we thought that its inclusion can be informative for evaluation of TVM. In addition, we also used the Self-Tuned Forgetron algorithm [5], 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 $m_1 = 1$, $m_2 = 2$, and $\eta = 0.2$, with *InputSpace*, *OneSided* and *Rejection* merging were used in all the experiments. 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 represents the model-free online learning approach and serves as a lower bound on the accuracy achievable by TVM.

We performed three set of experiments on 12 data sets with three different kernels: linear kernel $K(x, y) = xy$, RBF kernel $K(x, y) = \exp(\frac{-||x-y||^2}{2\delta})$, and polynomial kernel $K(x, y) = (xy + 1)^d$. Although different kernels might be appropriate for different data sets, our goal was to compare behavior of the SVM algorithms over a large range of conditions. To keep things simple, and consulting previous SVM evaluations [15], LIBSVM and IDSVM used the slack parameter $C = 1$ for *Adult* and $C = 100$ for the remaining 11 data sets. Similarly, following the strategy explained in section 4.3, TVM kept the product of C and the sum of twin weights at B for Adult and at $100B$ for the remaining 11 data sets. RBF kernel width δ was set to $\delta = K/2$ [3], where K is the number of attributes, for all data sets but the two *Checkerboard* data sets. There, the width was set to 0.37. Polynomial kernel degree d was set to 3 in all data



(a) 100 observed

(b) 1000 observed

(c) 10000 observed

(d) 10 million observed

(e) accuracy comparison

(f) time comparison

Figure 2: TVM(B=100) solutions on 10 million N-Checkerboard data.

sets instead the two *Checkerboard* data sets. There, the degree was set to $d = 7$ because of the complexity of the problem. For the Forgetron, RBF kernel width was differently tuned for better performance. All experiments were run on a 3G RAM, 3.2GHz Pentium Dual Core 2 PC.

5.3 TVM Trained on 10 Million N-Checkerboard Examples. In Figures 2.a-d we illustrate the TVM solutions with budget $B = 100$ and RBF kernel after $N = 100, 1000, 10000$, 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 Vectors. It can be seen that the TVM decision boundary dramatically improved during the process. This success can be attributed to the improved positioning of twin vectors near the decision boundary. It is interesting to observe that the TVM margin gradually decreased during the procedure. The solution for $N = 100$ (Figure 2.a) corresponds to Random and it is evident that TVM managed to substantially improve its accuracy by careful choice of twin vectors.

In Figure 2.e we compare accuracies of IDSVM, LIBSVM, and TVM with $B = 100$ as a function of the stream size. IDSVM was suspended after $N = 10,000$ examples and LIBSVM was trained on $N = 100,000$ 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 Figure 2.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 while the IDSVM runtime appears cubic, as expected.

5.4 Results on Benchmark Data Sets. In this section we summarize performance results on all 12 benchmark data sets. Each result listed in Tables 2, 3, 4 is an average of 5 repeated experiments. The first three columns of Table 2 compare the runtime of TVM with budget $B = 100$, LIBSVM, and IDSVM. The last three columns show number of support/twin vectors in the final TVM, LIBSVM, and IDSVM classifiers. To speed-up the experiments, we decided to terminate IDSVM after 3,000 seconds. In this case, in the last column we report the number of support vectors and how many training examples were processed by the time of stopping. The runtime of IDSVM was about two orders of magnitude larger than for LIBSVM. This difference was largely caused by the fact that IDSVM is

Table 2: The summary of training time and # of SVs/TVs with RBF kernel. (* means early stopped after 3,000 seconds. Superscript values indicate # of examples learned before early stopped.)

Data sets	Training time (in seconds)			# of SVs/TVs		
(RBF Kernel)	TVM	LIBSVM	IDSVM	TVM	LIBSVM	IDSVM
Adult	230	395	$\ast$	100	8578	3223^{7200}
Banana	21	1	54	100	960	1015
Checkerboard	1406	112	$\ast$	100	4073	3410^{7560}
N-Checkerboard	1778	3604	$\ast$	100	51230	5440^{10000}
Covertype	1709	13100	$\ast$	100	38790	2617^{5400}
Gauss	774	3940	$\ast$	100	39650	3212^{5400}
IJCNN	122	231	$\ast$	100	2103	1257^{23400}
Letter	138	5	$\ast$	100	2075	1261^{6800}
Pendigits	33	2	168	100	639	641
Shuttle	11	5	601	100	189	384
USPS	186	24	1621	100	1077	1081
Waveform	304	38673	$\ast$	100	24905	2497^{10000}

Table 3: Accuracy(×100%) comparison on 12 large data sets by RBF kernel.(Superscript values indicate # of training examples learned by IDSVM after 3,000 seconds.)

Data sets	Forgetron	Random	LIBSVM	IDSVM	TVM with different budgets				
(RBF)	B=100	B=100	$B = \infty$	$B = \infty$	B=20	B=50	B=100	B=200	B=500
Adult	80.7	79.5	84.3	84.0	83.0	82.9	82.1	82.8	83.7
Banana	83.7	86.9	90.2	91.6	87.5	88.3	89.8	89.8	89.9
Checkerboard	82.4	82.9	99.8	99.7^{7560}	75.0	91.3	98.1	98.4	99.0
N-Checkerboard	70.3	72.9	96.9	95.8^{10000}	73.3	91.4	97.1	97.7	98.8
Covertype	56.4	64.4	85.3	80.5^{5400}	66.9	71.4	76.5	77.9	79.6
Gauss	77.3	78.7	81.5	80.7^{5400}	81.3	81.3	81.1	81.4	81.8
IJCNN	88.1	90.3	98.7	98.4^{23400}	93.1	95.8	97.0	97.4	97.9
Letter	73.1	71.9	93.2	95.0^{6800}	76.5	82.5	87.6	90.3	92.0
Pendigits	96.6	93.9	99.5	98.7	96.6	98.7	99.1	99.1	99.2
Shuttle	99.4	98.3	99.9	99.9	99.7	99.6	99.8	99.8	99.8
USPS	83.9	86.8	96.1	96.7	85.0	88.5	92.1	93.5	95.0
Waveform	82.5	85.4	87.9	89.0^{10000}	86.9	86.1	87.7	88.5	88.8
(Average of all)	**81.2**	**82.7**	**92.8**	**92.5**	**83.7**	**88.2**	**90.7**	**91.4**	**92.1**

Table 4: Accuracy(×100%) comparison on 12 large data sets by polynomial & linear kernels. (Superscript values in IDSVM column indicate # of training examples learned by IDSVM after 3,000 seconds.)

	Polynomial				Linear			
Data sets	Forgetron	Random	IDSVM	TVM	Forgetron	Random	IDSVM	TVM
	B=100	B=100	$B{=}\infty$	B=100	B=100	B=100	$B{=}\infty$	B=100
Adult	71.8	78.1	72.4^{7200}	81.2	61.9	74.4	84.5^{8600}	81.3
Banana	61.6	76.2	76.3	83.0	48.8	55.1	54.2	56.4
Checkerboard	54.1	81.9	99.8^{67600}	91.1	49.8	48.7	49.7^{5400}	51.7
N-Checkerboard	49.2	71.6	69.9^{7600}	93.9	50.0	48.8	49.2^{3200}	50.8
Covertype	60.0	59.1	71.6^{6000}	73.9	59.4	66.9	75.7^{10400}	75.6
Gauss	71.4	78.6	80.9^{15400}	81.1	65.7	75.4	76.2^{14800}	76.9
IJCNN	84.1	88.0	95.1^{23800}	96.9	52.5	89.8	92.1	94.2
Letter	68.5	68.0	88.6	84.7	63.9	66.8	72.2^{11200}	72.5
Pendigits	95.0	93.2	98.3	98.8	82.6	85.9	90.7	90.1
Shuttle	93.7	95.9	99.9	99.6	86.6	95.9	98.1	98.1
USPS	83.3	86.6	95.8	92.8	75.4	78.7	86.6^{6600}	83.8
Waveform	75.9	78.3	81.6^{4600}	86.6	65.1	79.6	85.2^{17400}	85.3
(Average of all)	**72.4**	**79.6**	**85.9**	**88.6**	**63.5**	**72.2**	**76.2**	**76.4**

Table 5: Accuracy(×100%) comparison of different TVM versions. (Superscript values indicate # of training examples learned by IDSVM after 3,000 seconds; numbers in italics in TVM columns indicate that accuracy is more than 3% smaller than for the default TVM.)

Data sets	Random	IDSVM	TVM with different heuristics(B=100)					
(RBF Kernel)	B = 100	B = ∞	Default	[-1 1]	[-2 2]	NoRejection	Global	FeatureSpace
Adult	79.5	84.0	82.1	81.4	80.5	83.2	82.3	81.8
Banana	86.9	91.6	89.8	88.7	89.8	90.5	90.4	90.3
Checkerboard	82.9	99.7^{7560}	98.1	97.3	95.2	97.9	95.5	97.4
N-Checkerboard	72.9	95.8^{10000}	97.1	97.1	*89.1*	97.2	94.6	96.9
Covertype	64.4	80.5^{5400}	76.5	76.6	*72.7*	75.6	*70.3*	76.0
Gauss	78.7	80.7^{5400}	81.1	81.2	81.2	81.2	81.2	81.2
IJCNN	90.3	98.4^{23400}	97.0	97.3	*92.1*	*90.0*	95.9	96.8
Letter	71.9	95.0^{6800}	87.6	87.5	*78.4*	*78.8*	*80.4*	86.9
Pendigits	93.9	98.7	99.1	98.9	*93.0*	98.8	98.6	99.2
Shuttle	98.3	99.9	99.8	99.8	98.0	99.8	99.8	99.8
USPS	86.8	96.7	92.1	91.7	*84.3*	*81.9*	91.1	91.9
Waveform	85.4	89.0^{10000}	87.7	87.1	*83.2*	*84.3*	88.1	87.6
(Average of all)	**82.7**	**92.5**	**90.7**	**90.4**	**86.5**	**88.3**	**89.0**	**90.5**

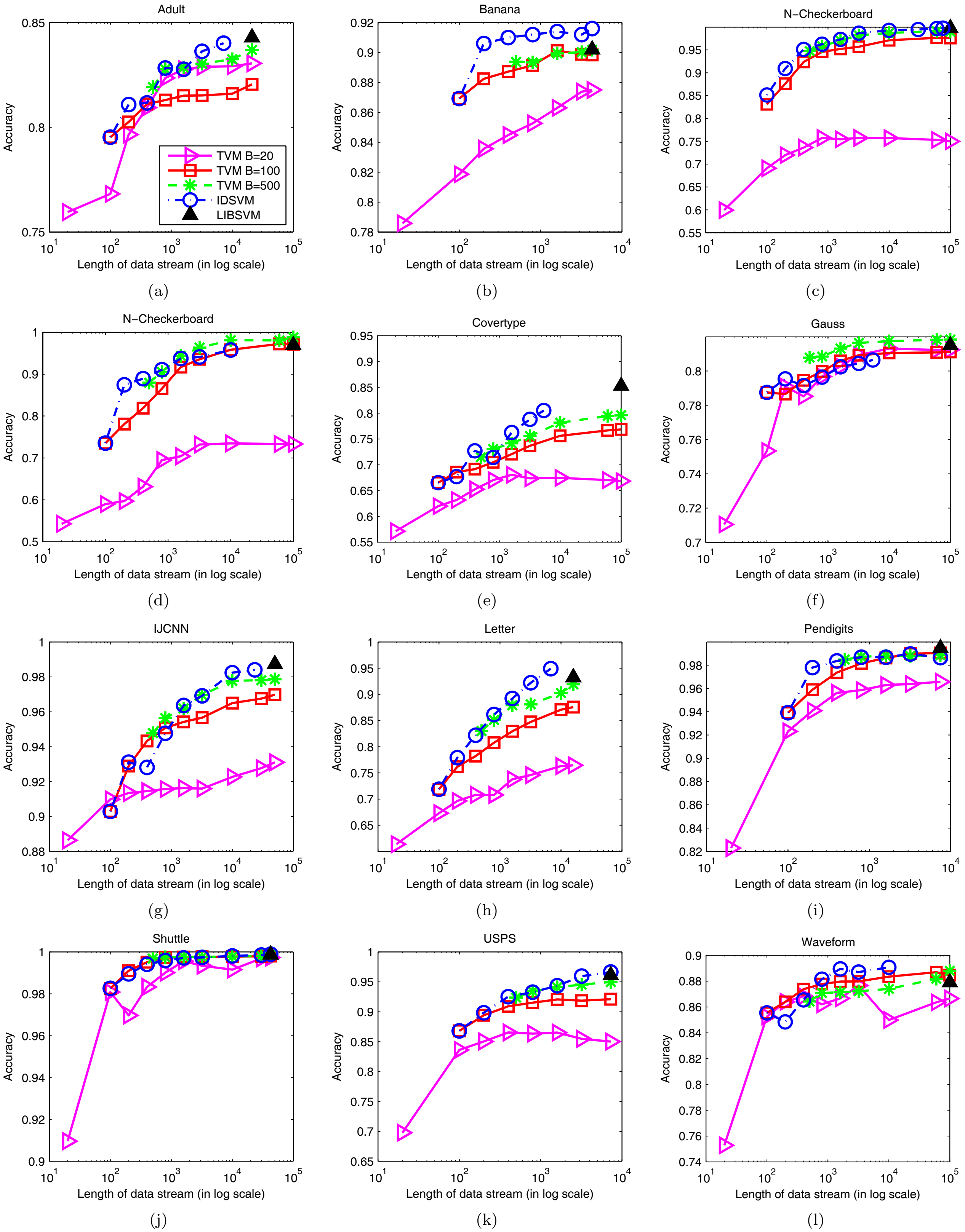
implemented in Matlab, while LIBSVM is implemented in C++. We note that TVM was also implemented in Matlab by borrowing many of the IDSVM routines. This explains why TVM was faster than LIBSVM on the largest data sets, despite its inferiority on the smallest data sets. On the other hand, TVM was clearly much faster than IDSVM on all data sets. Both LIBSVM and IDSVM classifiers had comparable number of support vectors that was in most cases much larger than the TVM's budget of 100 twin vectors.

In Table 3 we compare accuracies of the competing algorithms using RBF kernels, while in Figure 3 we illustrate how the accuracy of TVM and IDSVM depended on the number of observed training examples. As expected, the memory-unbounded LIBSVM and IDSVM had the highest accuracy. The two accuracies were quite similar and the small advantage of LIBSVM could probably be attributed to occasional early stopping of IDSVM. Importantly, TVM accuracies were very competitive and, in most cases, comparable to the memory-unbounded SVMs. TVM accuracies consistently increased with budget B. It is worth noting that TVM achieved impressive accuracies for a very modest budget of $B = 100$, while the 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 it. Interestingly, Forgetron with $B = 100$ was not impressive.

Let us briefly discuss some specific results from Table 3. TVM was the most accurate classifier on *N-Checkerboard* data. This can be explained by a very high noise level in the data, which TVM managed to control by ignoring the twin vectors outside the margin and by merging which revealed the class distribution. Similar behavior was observed on other two noisy data sets, *Gauss* and *Waveform*. In this case, TVM with a tiny budget of $B = 20$ was equally successful to LIBSVM that created tens of 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 and IDSVM. On *Covertype*, the difference was large even with $B = 500$, which is understandable considering that 40% of the 100,000 training examples were used as support vectors in LIBSVM.

A glance at Figure 3 reveals further details specific for each of the benchmark data sets. It reveals another strength of TVM: for every choice of budget B and every data set, 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 4 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 data sets we examined. Despite this, the relative differences between the 4 algorithms were consistent with the results in Table 3.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

Figure 3: The accuracy comparison on 12 large datasets. (Budget=20,100,500 for TVM)

Namely, TVM accuracy with $B = 100$ was quite comparable to 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.

5.5 Evaluation of TVM Default Parameters.

Table 5 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 was probably caused because $m_1 = 2$ prevented reduction in margin size. *Rejection* merging appeared highly successful because its omission (column 6) caused substantial drop in accuracy (it was especially large on *IJCNN*, *Letter*, and *USPS*). Similarly, omission of *One-Sided* merging (column 7) negatively impacted the accuracy. Finally, feature space merging (column 8) resulted in practically the same 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.

6 Conclusion

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 to the memory-unbounded SVM algorithms. This hints at the possibility of building accurate SVM classifiers from very 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. 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. Another question is related to the quadratic memory scaling of TVM with budget size. Kernel perceptrons are popular algorithms whose memory scales linearly with the budget. As seen, performance of the state of the art Forgetron algorithm was not impressive. It would be interesting to explore if some of the ideas used in TVM could improve performance of kernel perceptron algorithms. Finally, lossy data compression is another way of addressing the memory constraint. An open question is whether and how data compression could be integrated with data mining algorithms to further improve the computational footprint of kernel based supervised algorithms.

References

[1] G. Cauwenberghs and T. Poggio, Incremental and Decremental Support Vector Machine Learning, *NIPS*, 2000.

[2] N. Cesa-Bianchi and C. Gentile, Tracking the best hyperplane with a simple budget perceptron, *Annual Conference on Computational Learning Theory*, 2006.

[3] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines, 2001.

[4] K. Crammer and J. Kandola and Y. Singer, Online classification on a budget, *NIPS*, 2004.

[5] O. Dekel and S. S. Shwartz and Y. Singer, The Forgetron: A kernel-based Perceptron on a budget, *SIAM J. Comput.*, 2008.

[6] C. P. Diehl and G. Cauwenberghs, SVM incremental learning, adaptation and optimization, *IJCNN*, 2003.

[7] T. Hastie, S. Rosset, R. Tibshirani and J. Zhu, The entire regularization path for the support vector machine, *Journal of Machine Learning Research*, 2004.

[8] S. Haykin, Adaptive Filter Theory, Prentice Hall, 2002.

[9] D. Nguyen and T. Ho, An efficient method for simplifying support vector machines, *ICML*, 2005.

[10] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review*, 1958.

[11] G. Schohn and D. Cohn, Less is more: Active learning with support vector machines, *ICML*, 2000.

[12] S. Tong and D. Koller, Support vector machine active learning with applications to text classification, *ICML*, 2000.

[13] I. W. Tsang and J. T. Kwok and P.-M. Cheung, Core vector machines: Fast SVM training on very large data sets, *Journal of Machine Learning Research*, 2005.

[14] V. N. Vapnik, Statistical Learning Theory, John Wiley Sons, Inc., 1998

[15] S. V. N. Vishwanathan and A. J. Smola and M. N.Murty. SimpleSVM, *ICML*, 2003.

[16] J.S. Viter , Random sampling with a reservoir, *ACM transactions on mathematical software*, 1985.

[17] H. Yu and J. Yang and J. Han, Classifying large data sets using SVM with hierarchical clusters, *SIGKDD*, 2003.