Journal of Machine Learning Research ? (201?) ??-?? Submitted ??/??; Published ??/??

Breaking the Curse of Kernelization: Budgeted Stochastic Gradient Descent for Large-Scale SVM Training

Zhuang Wang[1] ZHUANG.WANG@SIEMENS.COM
Corporate Research and Technology
Siemens Corporation
755 College Road East
Princeton, NJ 08540, USA

Koby Crammer KOBY@EE.TECHNION.AC.IL
Department of Electrical Engineering
The Technion
Mayer Bldg
Haifa, 32000, Israel

Slobodan Vucetic VUCETIC@TEMPLE.EDU
Department of Computer and Information Sciences
Temple University
1805 N Broad Street
Philadelphia, PA 19122, USA

Editor: Tong Zhang

Abstract

Online algorithms that process one example at a time are advantageous when dealing with very large data or with data streams. Stochastic gradient descent (SGD) is such an algorithm and it is an attractive choice for online SVM training due to its simplicity and effectiveness. When equipped with kernel functions, similarly to other SVM learning algorithms, SGD is susceptible to "the curse of kernelization" that causes unbounded linear growth in model size and update time with data size. This may render SGD inapplicable to large data sets. We address this issue by presenting a class of Budgeted SGD (BSGD) algorithms for large-scale kernel SVM training which have *constant space and time complexity per update*. BSGD keeps the number of support vectors bounded during training through several budget maintenance strategies. We treat the budget maintenance as a source of the gradient error, and relate the gap between the BSGD and the optimal SVM solutions via the average model degradation due to budget maintenance. To minimize the gap, we study greedy budget maintenance methods based on removal, projection, and merging of support vectors. We propose budgeted versions of several popular online SVM algorithms that belong to the SGD family. We further derive BSGD algorithms for multi-class SVM training. Comprehensive empirical results show that BSGD achieves much higher accuracy than the state-of-the-art budgeted online algorithms and comparable to non-budget algorithms, while achieving impressive computational efficiency both in time and space during training and prediction.

Keywords: SVM, large-scale learning, online algorithms, online learning, stochastic gradient descent, kernel methods, learning with budget

[1] Zhuang Wang was with the Department of Computer and Information Sciences at Temple University while most of the presented research was performed.

1 Introduction

Computational complexity of machine learning algorithms becomes a limiting factor when one is faced with very large amounts of data. In an environment where new large scale problems are emerging in various disciplines and pervasive computing applications are becoming common, there is a real need for machine learning algorithms that are able to process increasing amounts of data efficiently. Recent advances in large-scale learning resulted in many algorithms for training Support Vector Machines (SVM) using large data (Vishwanathan et al., 2003; Zhang, 2004; Bordes et al., 2005; Tsang et al., 2005; Joachims, 2006; Hsieh et al., 2008; Bordes et al., 2009; Zhu et al., 2009; Teo et al., 2010; Chang et al., 2010; Sonnenburg and Franc, 2010; Yu et al., 2010; Shalev-Shwartz et al., 2011). However, while most of these algorithms focus on linear classification problems, the area of large-scale kernel SVM training remains less explored. SimpleSVM (Vishwanathan et al., 2003), LASVM (Bordes et al., 2005), CVM (Tsang et al., 2005) and parallel SVMs (e.g. Zhu et al., 2009) are among the few successful attempts to train kernel SVM from large data. However, these algorithms do not bound the model size and, as a result, they typically have quadratic training time in the number of training examples. This limits their practical use on very large-scale data sets.

A promising avenue to SVM training from large data sets and from data streams is to use online algorithms. Online algorithms operate by repetitively receiving a labeled example, adjusting the model parameters, and discarding the example. This is opposed to offline algorithms where the whole collection of training examples is at hand and training is accomplished by batch learning. Stochastic Gradient Decent (SGD) is a recently popularized approach (Shalev-Shwartz et al., 2011) that can be used for online training of SVM, where the objective is cast as an unconstrained optimization problem. Such algorithms proceed by iteratively receiving a labeled example and updating the model weights through gradient decent over the corresponding instantaneous objective function. It was shown that SGD converges toward the optimal SVM solution as the number of examples grows (Shalev-Shwartz et al., 2011). In its original non-kernelized form SGD has constant update time and constant space.

To solve nonlinear classification problems, SGD and related algorithms, including the original perceptron (Rosenblat, 1958), can be easily kernelized combined with Mercer kernels, resulting in prediction models that require storage of a subset of observed examples, called the Support Vectors (SVs). While kernelization allows solving highly nonlinear problems, it also introduces heavy computational burden. The main reason is that on noisy data the number of SVs tends to grow linearly with the number of training examples (Steinwart, 2003). In addition to the danger of exceeding the physical memory, this also implies a linear growth in both model update and prediction time with data size. We refer to this property of kernel online algorithms as *the curse of kernelization*. To solve the problem, budgeted online SVM algorithms (Crammer et al., 2004) that limit the number of support vectors were proposed to bound the number of SVs. In practice, the assigned budget depends on the specific application requirements, such as memory limitations, processing speed, or data throughput.

In this paper we study a class of BSGD algorithms for online training of kernel SVM. The main contributions of this paper are as follows. First, we propose a budgeted version of the kernelized SGD for SVM that has constant update time and constant space. This is achieved by controlling the number of SVs through one of the several budget maintenance strategies. We study the impact of budget maintenance on SGD optimization and show that, in the limit, the gap between the loss of BSGD and the loss of the optimal solution is upper-bounded by the average model degradation induced by budget maintenance. Second, we develop a multi-class version of BSGD based on the multi-class SVM formulation by Crammer & Singer (2001). The resulting multi-class BSGD has similar algorithmic structure as its binary relative and inherits its theoretical properties. Having shown that the quality of BSGD directly depends on the quality of budget maintenance, our final contribution is exploring computationally efficient methods to maintain an accurate low-budget classifier. In this work we consider three major budget maintenance strategies: removal, projection, and merging. In case of removal, we show that it is optimal to remove the smallest SV. Then, we show that optimal projection of one SV to the remaining ones is achieved by minimizing the accumulated loss of multiple sub-problems for each class, which extends the results by Csató & Opper (2001), Engel et al. (2002), and Orabona et al. (2009) to the multi-class setting. In case of merging, when Gaussian kernel is used, we show that the new SV is always on the line connecting two merged SVs, which generalizes the result by Nguyen & Ho (2005) to the multi-class setting. Both space and update time of BSGD scale quadratically with the budget size when projection is used and linearly when merging or removal are used. We show experimentally that BSGD with merging is very attractive because it is computationally efficient and results in highly accurate classifiers.

The structure of the paper is as follows: related work is given in section 2; a framework for the proposed algorithms is presented in section 3; the impact of budget maintenance on SGD optimization is studied in section 4, which motivates the budget maintenance strategies that are presented in section 6; the extension to the multi-class setting is described in section 5; in section 7, the proposed algorithms are comprehensively evaluated; and, finally, the paper is concluded in section 8.

2 Related Work

In this section we summarize related work to ours. Figure 1 summarizes an organization of large-scale SVM training algorithms discussed below.

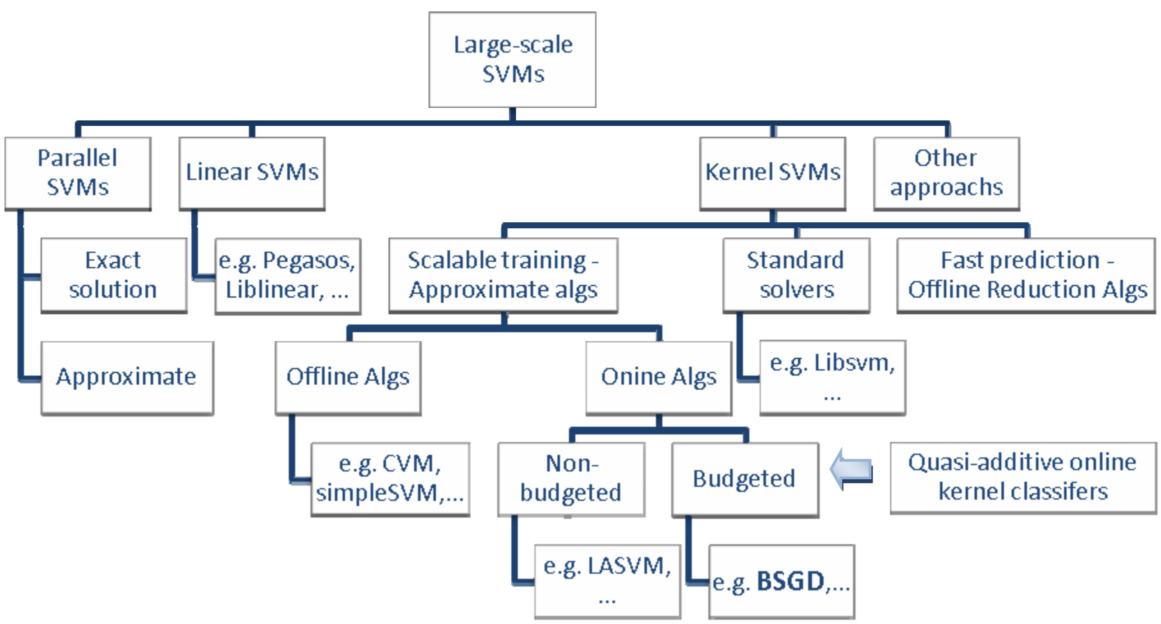


Figure 1. A hierarchy of large-scale SVMs

2.1 Algorithms for Large-Scale SVM Training

LIBSVM (Chang and Lin, 2001) is a widely used SVM solver which is scalable to hundreds of thousands of examples. LIBSVM uses the SMO decomposition technique (Platt, 1998) to solve SVM's Quadratic Programming (QP). LASVM (Bordes et al., 2005) is another scalable SMO-based algorithm that approximates the SVM solution by incrementally updating the model. In order to speed up training, LASVM performs only several SMO iterations during each model update and it occasionally removes examples from the training set that are deemed unlikely to become SVs. SimpleSVM (Vishwanathan et al., 2003) is a fast iterative training algorithm that uses greedy working set selection to identify SVs to be incrementally updated. CVM (Tsang et al., 2005) scales up kernel SVM by reformulating SVM's QP as a minimum enclosing ball problem and it applies an efficient approximation algorithm to obtain a near-optimal solution. SimplerCVM (Tsang et al., 2007) is a version that reduces the minimum enclosing ball problem to the enclosing ball problem and thus solves a simpler problem. Experimentally, these approximate algorithms have been demonstrated to have relatively fast training times, result in sparser models, and achieve a slightly reduced accuracy.

Recent research in large-scale linear SVM resulted in many successful algorithms (Zhang, 2004; Joachims, 2006; Shalev-Shwartz et al., 2011; Hsieh et al., 2008; Bordes et al., 2009; Teo et al., 2010; Chang et al., 2010) with an impressive scalability and able to train with millions of examples in a matter of minutes on standard PCs. Recently, linear SVM algorithms have been employed for nonlinear classification by explicitly expressing the feature space as a set of attributes and training a linear SVM on the transformed data set (Rahimi and Recht, 2007; Sonnenburg and Franc, 2010; Yu et al., 2010). However, this type of approaches is only applicable either with a limited type of kernels (e.g. the low degree polynomial kernels, string kernels or shift invariant kernels) or on very sparse or low dimensional data sets. Very recently, Zhang et al., (2012) proposed a low-rank linearization approach that is general to any PSD kernel. The proposed algorithm LLSVM transforms a non-linear SVM to a linear one via an approximate empirical kernel map computed from low-rank approximation of kernel matrices. Taking the advantage of the fast training of linear classifiers, Wang et al. (2011) proposed to use multiple linear classifiers to capture non-linear concepts. A common property of the above *linear-classifier-based* algorithms is that they usually have low space footprint and are initially designed for offline learning but can also be easily adapted to online algorithms by compromising a certain degree of accuracy. Recent research in training large-scale SVM with the popular Gaussian kernel focuses on parallelizing training on multiple cores or machines. Either optimal (e.g. Graf et al., 2005) or approximate (e.g. Zhu et al., 2009) solutions can be obtained by this type of methods. Other attempts to large-scale kernel SVM learning include the method of modifying the loss function in the SVM optimization problem (Collobert et al., 2006), preprocessing methods such as pre-clustering and training on the high-quality summarized data (Li et al., 2007), and the method (Chang et al., 2010) that decomposes data space and train SVMs on the decomposed regions.

2.2 Algorithms for SVM Model Reduction

SVM model can be thought of as composed of a subset of training examples known as SVs, whose number typically grows linearly with the number of training examples on noisy data (Steinwart, 2003). Bounding the space complexity of SVM classifiers has been an active research since the early days of SVM. SVM reduced set methods (Burges, 1996; Schökopf et al., 1999) start by training a standard SVM on the complete data and then find a sparse approximation by minimizing Euclidean distance between the original and the approximated SVM. A limitation of reduced set methods is that they require training a full-scale SVM, which can be computationally infeasible on large data. Another line of work (Lee and Mangasarian 2001; Wu et al., 2005; Dekel and Singer, 2006) is to directly train a reduced classifier from scratch by reformulating the optimization problem. The basic idea is to train SVM with minimal risk on the complete data under a constraint that the model weights are spanned by a small number of examples. A similar method to build reduced SVM classifier based on forward selection was proposed by Keerthi et al. (2006). This method proceeds in an iterative fashion that greedily selects an example to be added to the model so that the risk on the complete data is decreased the most. Although SVM reduction methods can generate a classifier with a fixed size, they require multiple passes over training data. As such, they can be infeasible for online learning.

2.3 Online Algorithms for SVM

Online SVM algorithms were proposed to incrementally update the model weights upon receiving a single example. IDSVM (Cauwenberghs and Poggio, 2001) maintains the optimal SVM solution on all previously seen examples throughout the whole training process by using matrix manipulation to incrementally update the KKT conditions. The high computational cost due to the desire to guarantee an optimum makes it less practical for large-scale learning. As an alternative, LASVM (Bordes et al., 2005) was proposed to trade the optimality with scalability by using an SMO like procedure to incrementally update the model. However, LASVM still does not bound the number of SVs and a potential unlimited growth in their number limits its use for truly large learning tasks. Both IDSVM and LASVM solve SVM optimization by casting it as a QP problem and working on the KKT conditions.

Gradient-based methods are an appealing alternative to the QP based methods for SVM training. Stochastic gradient decent for SVM training was first studied by Kivinen et al. (2002), where SVM training is cast as an unconstrained problem and model weights are updated through gradient decent over an instantaneous objective function. Pegasos (Shalev-Shwartz et al., 2011) is an improved stochastic gradient method, by employing a more aggressively decreasing learning rate and projection. Iterative nature of stochastic gradient makes it suitable for online SVM training. In practice, it is often run in epochs, by scanning the data several times to achieve a convergence to the optimal solution. Recently, Bordes et al. (2009) explored the use of 2nd order information to calculate the gradient in the SGD algorithms. Although the SGD-based methods show impressive training speed for linear SVMs, when equipped with kernel functions, they suffer from the curse of kernelization.

TVM (Wang and Vucetic, 2010a) is a recently proposed budgeted online SVM algorithm which has constant update time and constant space. The basic idea of TVM is to upper bound the number of SVs during the whole learning process. Examples kept in memory (called prototypes) are used both as SVs and as summaries of local data distribution. This has been achieved by positioning the prototypes near the decision boundary, which is the most informative region of the input space. An optimal SVM solution is guaranteed over the set of prototypes at any time. Upon removal or addition of a prototype, the IDSVM is employed to update its model.

2.4 Budgeted Quasi-additive Online Algorithms

The Perceptron (Rosenblatt, 1958) is a well-known online algorithm which is updated by simply adding misclassified examples to the model weights. Perceptron belongs to a wider class of quasi-additive online algorithms that updates a model in a greedy manner by using only the last observed example. Popular recent members of this family of algorithms include ALMA (Gentile, 2001), ROMMA (Li and Long, 2002), MIRA (Crammer and Singer, 2003), PA (Crammer et al., 2006), ILK (Cheng et al., 2007), the SGD based algorithms (Kivinen et al., 2002; Zhang, 2004; Shalev-Shwartz et al., 2011), and the Greedy Projection algorithm (Zinkevich, 2003). These algorithms are straightforwardly kernelized. To prevent the curse of kernelization, budget maintenance strategies for the kernel perceptron have been proposed in much recent work. The common constraint of the methods summarized below is that the number of SVs (the budget) is fixed to a pre-specified value.

Stoptron is a truncated version of kernel perceptron that terminates when number of SVs reaches budget B. This simple algorithm is useful for benchmarking (Orabona et al., 2009).

Budget Perceptron (Crammer et al., 2004) removes an SV that would be predicted correctly and with the largest confidence after its removal is selected for removal. While this algorithm performs well on relatively noise-free data it is less successful on noisy data. This is because in the noisy case this algorithm tends to remove well-classified points and accumulate noisy examples, resulting in a gradual degradation of accuracy.

Random Perceptron employs a simple removal procedure that removes a random SV. Despite its simplicity, this algorithm often has satisfactory performance and its convergence has been proven under some mild assumptions (Cesa-Bianchi and Gentile, 2006).

Forgetron uses another simple approach, inspired by the Forgetron algorithm (Dekel et al., 2008) by removing the oldest SV. The intuition is that the oldest SV was created when the quality of perceptron was the lowest and that its removal would be the least hurtful. After each update step, a forgetting factor is used to scale the current model and all its SVs. The oldest SV (the one with the smallest weight) is removed if the budget is exceeded. Under some mild assumptions, convergence of the algorithm has also been proven.

It is worth mentioning that a unified analysis of the convergence of Random Perceptron and Forgetron under the framework of online convex programming was studied by Sutskever (2009) after slightly modifying the two original algorithms.

Tighter Perceptron. The budget maintenance strategy proposed by Weston et al. (2005) is to evaluate accuracy on validation data when deciding which SV to remove. Specifically, the SV whose removal would have the least validation error is selected for removal. From the perspective of accuracy estimation, it is ideal that the validation set consists of all observed examples. However, such validation set implies an unbounded growth in space and time, and breaks the budget requirement. The budget alternative is to use the SV set for validation, but its drawback is that the SV set is not a representative of the underlying distribution, which in turn results in poor accuracy estimation.

Tightest Perceptron is a modification of Tighter Perceptron that uses the SV set both for model representation and accuracy estimation (Wang and Vucetic, 2009). This has been achieved by using SVs to approximate distribution of training examples and utilize this approximation in validation.

Projectron maintains a sparse representation by occasionally projecting SV onto remaining SVs (Orabona et al., 2009). The projection is designed to minimize the model weight degradation caused by removal of an SV, which requires updating the weights of the remaining SVs. The original algorithm does not enforce a fixed budget, but adaptively increases it according to a predefined sparsity parameter. It can be easily converted to the budgeted version by enforcing projection when the budget is exceeded.

SILK is the budgeted version of the ILK algorithm (Cheng et al., 2007). Once the buffer limit is exceeded, the example with the lowest absolute coefficient value is discarded to maintain a bound on memory usage.

BPA. Unlike the previously described algorithms that decouple model updating and budget maintenance, Wang and Vucetic (2010) proposed a budgeted online Passive-Aggressive (PA) algorithm (Crammer et al., 2006) that achieves a fixed budget by introducing an additional constraint into the original PA optimization problem. The constraint enforces that the removed SV is projected onto the space spanned by a subset of the remaining SVs. The optimization leads to a closed-form solution that jointly addresses model update and SV removal.

The properties of budgeted online algorithms described in this subsection and our proposed BSGD algorithms are summarized in Table 1. It is worth noting that although (budgeted) online algorithms are typically trained by a single pass through training data, they are also able to perform multiple passes that can lead to improved accuracy.

Algorithms	Budget maintenance	Update time[2]	Computational space
BPA_S	projection	$O(B)$	$O(B)$
BPA_{NN}	projection	$O(B)$	$O(B)$
BPA_P	projection	$O(B^2)$	$O(B^2)$
BSGD+removal	removal	$O(B)$	$O(B)$
BSGD+project	projection	$O(B^2)$	$O(B^2)$
BSGD+merge	merging	$O(B)$	$O(B)$
Budget	removal	$O(B)$	$O(B)$
Forgetron	removal	$O(B)$	$O(B)$
Projectron++	projection	$O(B^2)$	$O(B^2)$
Random	removal	$O(B)$	$O(B)$
SILK	removal	$O(B)$	$O(B)$
Stoptron	stop	$O(1)$	$O(B)$
Tighter	removal	$O(B^2)$	$O(B)$
Tightest	removal	$O(B^2)$	$O(B)$
TVM	merging	$O(B^2)$	$O(B^2)$

Table 1. Comparison of different budgeted online algorithms (B is a pre-specified number of SVs).

3 Budgeted Stochastic Gradient Descent (BSGD) for SVMs

In this section, we describe an algorithmic framework of BSGD for SVM training.

3.1 Stochastic Gradient Descent (SGD) for SVMs

Consider a binary classification problem with a sequence of labeled examples $S = \{(\mathbf{x}_i, y_i), i = 1,...,N\}$, where instance $\mathbf{x}_i \in \mathbb{R}^d$ is a d-dimensional input vector and $y_i \in \{+1,-1\}$ is the label. Training an SVM classifier[3] $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ using S, where $\mathbf{w}$ is a vector of weights associated with each input, is formulated as solving the following optimization problem

$$\min P(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{t=1}^{N} l(\mathbf{w};(\mathbf{x}_t, y_t)), \tag{3.1}$$

where $l(\mathbf{w};(\mathbf{x}_t, y_t)) = \max(0, 1 - y_t\mathbf{w}^T\mathbf{x}_t)$ is the *hinge loss* function and $\lambda > 0$ is a regularization parameter used to control model complexity.

The SGD algorithm is used to solve problem (3.1). SGD works iteratively. It starts with an initial guess of the model weight $\mathbf{w}_1$, and at t-th round it updates the current weight $\mathbf{w}_t$ as

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla_t, \tag{3.2}$$

where $\nabla_t = \nabla_{\mathbf{w}_t} P_t(\mathbf{w}_t)$ is the (sub)gradient of the instantaneous objective function

$$P_t(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + l(\mathbf{w};(\mathbf{x}_t, y_t)) \tag{3.3}$$

at $\mathbf{w}_t$, and $\eta_t > 0$ is a learning rate. Thus, (3.2) can be rewritten as

$$\mathbf{w}_{t+1} \leftarrow (1 - \lambda\eta_t)\mathbf{w}_t + \beta_t\mathbf{x}_t, \tag{3.4}$$

where

$$\beta_t \leftarrow \begin{cases} \eta_t y_t & \text{if } y_t\mathbf{w}_t^T\mathbf{x}_t < 1 \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

[2] Here, update time includes both model update time and budget maintenance time.

[3] We study the case where the bias term is set to zero.

Several learning algorithms are based on (or can be viewed as) SGD for SVM. In Table 2, Pegasos[4] (Shalev-Shwartz et al., 2011), Norma (Kivinen et al., 2002) and Margin Perceptron[5] (Duda and Hart, 1973) are viewed as the SGD algorithms. They share a unified update rule (3.4), but can differ in learning rate scheduling or, in the case of Margin Perceptron, by removing the regularization term.

Algorithms	λ	η_t
Pegasos	>0	$1/(\lambda t)$
Norma	>0	$\eta/\sqrt{t}$
Margin Perceptron	0	η

Table 2. A summary of three SGD algorithms (η is a constant.)

3.2 Kernelization

SGD for SVM can be used to solve non-linear problems when combined with Mercer kernels. After introducing a nonlinear function Φ that maps $\mathbf{x}$ from the input to the feature space and replacing $\mathbf{x}$ with $\Phi(\mathbf{x})$, $\mathbf{w}_t$ can be described as

$$\mathbf{w}_t = \sum\nolimits_{j=1}^{t} \alpha_j \Phi(\mathbf{x}_j), \tag{3.6}$$

where

$$\alpha_j = \beta_j \prod_{k=j+1}^{t} (1-\eta_k \lambda). \tag{3.7}$$

From (3.7) it can be seen that an example ($\mathbf{x}_t$, y_t) whose hinge loss was zero at time t has zero α coefficient and can therefore be ignored. Examples with nonzero α are called the Support Vectors[6] (SVs). We can now represent $f_t(\mathbf{x})$ as the kernel expansion

$$f_t(\mathbf{x}) = \mathbf{w}_t^T \Phi(\mathbf{x}) = \sum\nolimits_{j \in I_t} \alpha_j k(\mathbf{x}_j, \mathbf{x}), \tag{3.8}$$

where k s the Mercer kernel induced by Φ and I_t is the set of indexes of all SVs in $\mathbf{w}_t$. Rather than explicitly calculating $\mathbf{w}$ by using $\Phi(\mathbf{x})$, that might be infinite-dimensional, it is more efficient to save SVs to implicitly represent $\mathbf{w}$ and to use k when calculating the prediction $\mathbf{w}^T\Phi(\mathbf{x})$. This is known as the *kernel trick*. Therefore, an SVM model is completely described by either a weight vector $\mathbf{w}$ or by an SV set $\{(\alpha_i, \mathbf{x}_i), i \in I_t\}$. From now on, depending on the convenience of presentation, we will use either the $\mathbf{w}$ notation or the α notation interchangeably.

3.3 Budgeted SGD (BSGD)

To maintain a fixed number of SVs, the BSGD executes a budget maintenance step whenever the number of SVs exceeds a pre-defined budget B (i.e. $|I_{t+1}| > B$). It reduces the size of I_{t+1} by one, such that $\mathbf{w}_{t+1}$ is only spanned by B SVs. This results in degradation of the SVM model. We present a generic BSGD algorithm for SVM in Algorithm 1. Here, we denote by Δ_t the weight degradation caused by budget maintenance at t-th round, which is defined[7] as the difference between model weights before and after budget maintenance (Line 7 of Algorithm 1). We note that all

[4] In this paper we study the Pegasos algorithm without the optional projecting step (Shalev-Shwartz, Singer & Srebro, 2011). It is worth to note that we can have similar analysis for both cases (with or without the optional projecting step). We focus on this version since it has close connection to other two algorithms we study.

[5] Margin Perceptron is a robust variant of the classical perceptron (Rosenblatt, 1958), by changing the update criteria from $y\mathbf{w}^T\mathbf{x}<0$ to $y\mathbf{w}^T\mathbf{x}<1$.

[6] In this paper Support Vectors refer to the examples that contribute to the online classifier at any stage of online learning, which overloads the terminology of standard Support Vectors that refer to the examples with non-zero coefficients in the dual form of the final classifier.

[7] The formal definition for different strategies is presented in Algorithm 2.

budget maintenance strategies mentioned in Section 2.4, except BPA, can be represented as Line 7 of Algorithm 1.

Budget maintenance (Line 7) is a critical design issue. We describe below several budget maintenance strategies for BSGDs in Section 6. We motivate different strategies by studying in the next section how budget maintenance influences the performance of SGD.

Algorithm 1: BSGD
Input: data S, kernel k, regularization parameter λ, budget B;
Initialize: $b = 0$, $\mathbf{w}_1 = \mathbf{0}$;
0. **for** $t = 1,2,\ldots$
1. receive $(\mathbf{x}_t, y_t)$;
2. $\mathbf{w}_{t+1} \leftarrow (1-\eta_t\lambda)\mathbf{w}_t$;
3. **if** $l(\mathbf{w}_t;(\mathbf{x}_t, y_t)) > 0$
4. $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_{t+1} + \Phi(\mathbf{x}_t)\beta_t$; // add an SV
5. $b = b+1$;
6. **if** $b > B$
7. $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_{t+1} - \Delta_t$; // budget maintenance
8. $b = b-1$;
Output: $f_{t+1}(\mathbf{x})$

4 Impact of Budget Maintenance on SGD

This section describes the impact of budget maintenance on SGD which provide important guidelines for the design of budget maintenance strategies.

In the following we quantify the optimization error introduced by budget maintenance on three known SGD algorithms. Without loss of generality, we assume $\|\Phi(\mathbf{x})\| \le 1$.

First, we quantify the quality degradation of SGD optimization for Budgeted Pegasos (BPegasos), a BSGD algorithm using the Pegasos style learning rate from Table 2.

Theorem 1 *Let us consider BPegasos (Algorithm 1 using the Pegasos learning rate, see Table 2) running on a sequence of examples S. Let* $\mathbf{w}^*$ *be the optimal solution of problem (3.1). Define the gradient error* $E_t = \Delta_t / \eta_t$ *and assume* $\|E_t\| \le 1$. *Let*

$$U = \begin{cases} 2/\lambda & \text{if } \lambda \le 4 \\ 1/\sqrt{\lambda} & \text{otherwise.} \end{cases} \tag{4.1}$$

Then, the following inequality holds,

$$\frac{1}{N}\sum_{t=1}^{N} P_t(\mathbf{w}_t) - \frac{1}{N}\sum_{t=1}^{N} P_t(\mathbf{w}^*) \le \frac{(\lambda U + 2)^2(\ln(N)+1)}{2\lambda N} + 2U\overline{E}, \tag{4.2}$$

where $\overline{E} = \sum_{t=1}^{N}\|E_t\|/N$ *is defined as the average gradient error.*

The proof is in Appendix A.

Remarks on Theorem 1

- In Theorem 1 we quantify how budget maintenance impacts the quality of SGD optimization. Observe that as N grows, the second term in the right side of inequalities (4.2) converges to zero. Therefore, the averaged instantaneous loss of BSGD converges toward the averaged instantaneous loss of optimal solution $\mathbf{w}^*$, and the gap between these two is upper bounded by the averaged gradient error $\overline{E}$. *The results suggest that an optimal budget maintenance should attempt to minimize* $\overline{E}$. To minimize $\overline{E}$ in the setting of online learning, we propose a greedy procedure that minimizes $\|E_t\|$ at each round.
- The assumption $\|E_t\| \le 1$ can be easily held. Let us assume the *removal*-based budget maintenance method, where, at round t, SV with index t' is removed. Then, the weight degradation

is $\Delta_t = \alpha_{t'}\Phi(\mathbf{x}_{t'})$. By using (3.7) it can be seen that $\| E_t \|$ is not larger than 1,

$$\| E_t \| \le \left\| \frac{\alpha_{t'}}{\eta_t} \right\| = \lambda t \left\{ \eta_{t'} \prod_{j=t'+1}^{t} (1-\eta_j \lambda) \right\} = \lambda t \left\{ \eta_{t'} \cdot \frac{t'}{t'+1} \cdot \frac{t'+1}{t'+2} \cdot \ldots \cdot \frac{t-2}{t-1} \cdot \frac{t-1}{t} \right\} = 1.$$

Since our proposed budget maintenance strategy is to minimize $\| E_t \|$ at each round, so $\| E_t \| \le 1$ also holds.

Next, we show a similar theorem for Budgeted Norma (BNorma), a BSGD algorithm using the Norma style update rule from Table 1.

Theorem 2 *Let us consider BNorma (Algorithm 1 using the Norma learning rate from Table 2) running on a sequence of examples S. Let* $\mathbf{w}^*$ *be the optimal solution of problem (3.1). Assume* $\| E_t \| \le 1$. *Let U be defined as in (4.1). Then, the following inequality holds,*

$$\frac{1}{N}\sum_{t=1}^{N} P_t(\mathbf{w}_t) - \frac{1}{N}\sum_{t=1}^{N} P_t(\mathbf{w}^*) \le \frac{(2U^2/\eta + \eta(\lambda U + 2)^2)\sqrt{N}}{N} + 2U\bar{E}. \tag{4.3}$$

The proof is in Appendix B. The remarks on Theorem 1 also hold[8] for Theorem 2.

Next, we show the result for Budgeted Margin Perceptron (BMP). The update rule of Margin Perceptron (MP) summarized in Table 2 does not bound growth of the weight vector. To achieve a unified study, a projection step is added to MP after the SGD update to guarantee an upper bound on the norm of the weight vector[9]. More specifically, the new update rule is

$$\mathbf{w}_{t+1} \leftarrow \prod_C (\mathbf{w}_t - \nabla_t) \equiv \phi_t(\mathbf{w}_t - \nabla_t),$$

where C is the closed convex set with radius U and $\prod_C(\mathbf{u})$ defines the closest point to $\mathbf{u}$ in C. We can replace the projection operator with $\phi_t = \min\{1, U/\| \mathbf{w}_t - \nabla_t \|\}$. It is worth to note that, although the MP with the projection step solves an un-regularized SVM problem (i.e. $\lambda = 0$ in (3.1)), the projection to a ball with radius U does introduce the regularization by enforcing the weight with bounded norm U. The vector U should be treated as a hyper-parameter and smaller U values enforce simpler models.

After this modification, the resulting BMP algorithm can be described with Algorithm 1, where an additional projection step $\mathbf{w}_{t+1} \leftarrow \prod_C(\mathbf{w}_{t+1})$ is added at the end of each iteration (after Line 8 of Algorithm 1).

Theorem 3 *Let C be a closed convex set with a pre-specified radius U. Let BMP (Algorithm 1 using the PMP learning rate and the projection step) run on a sequence of examples S. Let* $\mathbf{w}^*$ *be the optimal solution to the problem (3.1) with* $\lambda = 0$ *and subject to the constraint* $\| \mathbf{w}^* \| \le U$. *Assume* $\| E_t \| \le 1$. *Then, the following inequality holds,*

$$\frac{1}{N}\sum_{t=1}^{N} P_t(\mathbf{w}_t) - \frac{1}{N}\sum_{t=1}^{N} P_t(\mathbf{w}^*) \le \frac{2U^2}{N\eta} + 2\eta + 2U\bar{E} \tag{4.4}$$

The proof is in Appendix C. The remarks on Theorem 1 also hold for Theorem 3.

5 BSGD for Multi-Class SVM

Algorithm 1 can be extended to the multi-class setting. In this section we show that the resulting multi-class BSGD inherits the same algorithmic structure and theoretical properties of its binary counterpart.

Consider a sequence of examples $S = \{(\mathbf{x}_i, y_i), i = 1,\ldots,N\}$, where instance $\mathbf{x}_i \in \mathbb{R}^d$ is a d-dimensional input vector and the multi-class label y_i belongs to the set $Y = \{1,\ldots,c\}$. We consider

[8] The assumption $\| E_t \| \le 1$ *holds when budget maintenance is achieved by removing the smallest SV, i.e.* $t' = \arg\min_{j \in I_{t+1}} \| \alpha_j \Phi(\mathbf{x}_j) \|$.

[9] The projection step (Zinkevich, 2003; Shalev-Shwartz et al., 2007; Sutskever, 2009) is a widely used technical operation needed for the convergence analysis.

the multi-class SVM formulation by Crammer & Singer (2001). Let us define the multi-class model $f^M(\mathbf{x})$ as

$$f^M(\mathbf{x}) = \arg\max_{i \in Y}\{f^{(i)}(\mathbf{x})\} = \arg\max_{i \in Y}\{(\mathbf{w}^{(i)})^T \mathbf{x}\},$$

where $f^{(i)}$ is the *i*-th class-specific predictor and $\mathbf{w}^{(i)}$ is its corresponding weight vector. By adding all class-specific weight vectors, we construct $\mathbf{W} = [\mathbf{w}^{(1)} \ldots \mathbf{w}^{(c)}]$ as the $d \times c$ weight matrix of $f^M(\mathbf{x})$. The predicted label of $\mathbf{x}$ is the class of the weight vector that achieves the maximal value $(\mathbf{w}^{(i)})^T \mathbf{x}$. Given this setup, training a multi-class SVM on S consists of solving the optimization problem

$$\min_{\mathbf{W}} P^M(\mathbf{W}) = \frac{\lambda}{2}\|\mathbf{W}\|^2 + \frac{1}{N}\sum_{t=1}^{N} l^M(\mathbf{W};(\mathbf{x}_t, y_t)), \tag{5.1}$$

where the binary hinge loss is replaced with the *multi-class hinge loss* defined as

$$l^M(\mathbf{W};(\mathbf{x}_t, y_t)) = \max(0, 1 + f^{(r_t)}(\mathbf{x}_t) - f^{(y_t)}(\mathbf{x}_t)), \tag{5.2}$$

where $r_t = \arg\max_{i \in Y, i \neq y_t} f^{(i)}(\mathbf{x}_t)$, and the norm of the weight matrix $\mathbf{W}$ is

$$\|\mathbf{W}\|^2 = \sum_{i \in Y}\|\mathbf{w}^{(i)}\|^2.$$

The subgradient matrix ∇_t of the multi-class instantaneous loss function,

$$P_t^M(\mathbf{W}) = \frac{\lambda}{2}\|\mathbf{W}\|^2 + l^M(\mathbf{W};(\mathbf{x}_t, y_t)),$$

at $\mathbf{W}_t$ is defined as $\nabla_t = [\nabla_t^{(1)} \ldots \nabla_t^{(c)}]$, where $\nabla_t^{(i)} = \nabla_{\mathbf{w}^{(i)}} P_t^M(\mathbf{W})$ is a column vector. If loss (5.1) is equal to zero then $\nabla_t^{(i)} = \lambda \mathbf{w}_t^{(i)}$. If loss (5.2) is above zero, then

$$\nabla_t^{(i)} = \begin{cases} \lambda \mathbf{w}_t^{(i)} - \mathbf{x}_t, & \text{if } i = y_t \\ \lambda \mathbf{w}_t^{(i)} + \mathbf{x}_t, & \text{if } i = r_t \\ \lambda \mathbf{w}_t^{(i)}, & \text{otherwise.} \end{cases}$$

Thus, the update rule for the multi-class SVM becomes

$$\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t - \eta_t \nabla_t = (1 - \eta_t \lambda)\mathbf{W}_t + \mathbf{x}_t \boldsymbol{\beta}_t,$$

where $\boldsymbol{\beta}_t$ is a row vector, $\boldsymbol{\beta}_t = [\beta_t^{(1)} \ldots \beta_t^{(c)}]$. If loss (5.1) is equal to zero, then $\boldsymbol{\beta} = \mathbf{0}$; otherwise,

$$\beta_t^{(i)} = \begin{cases} \eta_t, & \text{if } i = y_t \\ -\eta_t, & \text{if } i = r_t \\ 0, & \text{otherwise.} \end{cases}$$

When used in conjunction with kernel, $\mathbf{w}_t^{(i)}$ can be described as

$$\mathbf{w}_t^{(i)} = \sum_{j=1}^{t} \alpha_j^{(i)} \Phi(\mathbf{x}_j),$$

where

$$\alpha_j^{(i)} = \beta_j^{(i)} \prod_{k=j+1}^{t} (1 - \eta_k \lambda).$$

The budget maintenance step can be achieved as

$$\mathbf{W}_{t+1} \leftarrow \mathbf{W}_{t+1} - \boldsymbol{\Delta}_t \Rightarrow \mathbf{w}_{t+1}^{(i)} \leftarrow \mathbf{w}_{t+1}^{(i)} - \Delta_t^{(i)},$$

where $\boldsymbol{\Delta}_t = [\Delta_t^{(1)} \ldots \Delta_t^{(c)}]$ and the column vectors $\Delta_t^{(i)}$ are the coefficients for the *i*-th class-specific weight, such that $\mathbf{w}^{(i)}_{t+1}$ is spanned only by B SVs.

Algorithm 1 can be applied to the multi-class version after replacing scalar β_t with vector $\boldsymbol{\beta}_t$, vector $\mathbf{w}_t$ with matrix $\mathbf{W}_t$ and vector Δ_t with matrix $\boldsymbol{\Delta}_t$.

The analysis of the quality gap between BSGD and SGD optimization for the multi-class version is similar to that provided for its binary version, presented in Section 4. If we assume $\|\Phi(\mathbf{x})\|^2 \leq 1/2$, then the resulting multi-class counterparts of Theorems 1-3 become identical to their binary variants by simply replacing the text “problem (3.1)” with “problem (5.1)”.

6 Budget Maintenance Strategies

The analysis in Sections 4 and 5 indicates that budget maintenance should attempt to minimize the averaged gradient error $\bar{E}$. To minimize $\bar{E}$ in the setting of online learning, we propose a greedy procedure that minimizes the gradient error $\| E_t \|$ at each round. From the definition of $\| E_t \|$ in Theorem 1, minimizing $\| E_t \|$ is equivalent to minimizing the weight degradation $\| \mathbf{\Delta}_t \|$,

$$\min \| \mathbf{\Delta}_t \|^2 . \tag{6.1}$$

In the following, we address the problem (6.1) through three budget maintenance strategies: removing, projecting and merging of SV(s). We discuss our solutions under the multi-class setting, and consider the binary setting as a special case. Three styles of budget maintenance update rules are summarized in Algorithm 2 and discussed in more detail in the following 3 subsections.

Algorithm 2 Budget maintenance

- Removal:
  1. select some p;
  2. $\mathbf{\Delta}_t = \Phi(\mathbf{x}_p)\boldsymbol{\alpha}_p$;
- Projection:
  1. select some p;
  2. $\mathbf{\Delta}_t = \Phi(\mathbf{x}_p)\boldsymbol{\alpha}_p - \sum_{j \in I_{t+1} - p} \Phi(\mathbf{x}_j)\Delta\boldsymbol{\alpha}_j$;
- Merging:
  1. Select some m and n;
  2. $\mathbf{\Delta}_t = \Phi(\mathbf{x}_m)\boldsymbol{\alpha}_m + \Phi(\mathbf{x}_n)\boldsymbol{\alpha}_n - \Phi(\mathbf{z})\boldsymbol{\alpha}_z$;

6.1 Budget Maintenance through Removal

If budget maintenance removes the p-th SV, then

$$\mathbf{\Delta}_t = \Phi(\mathbf{x}_p)\boldsymbol{\alpha}_p ,$$

where the row vector $\boldsymbol{\alpha}_p = [\alpha_p^{(1)} \ldots \alpha_p^{(c)}]$ contains the c class-specific coefficients of j-th SV. The optimal solution of (6.1) is removal of SV with the smallest norm,

$$p = \arg\min_{j \in I_{t+1}} \| \boldsymbol{\alpha}_j \|^2 \, k(\mathbf{x}_j, \mathbf{x}_j).$$

Let us consider the class of translation invariant kernels where $k(\mathbf{x}, \mathbf{x}') = \tilde{k}(\mathbf{x} - \mathbf{x}')$, which encompasses the Gaussian kernel. Let us assume, without loss of generality, that $k(\mathbf{x}, \mathbf{x}) = 1$. In this case, the best SV to remove is the one with the smallest $\|\boldsymbol{\alpha}_p\|$. Note:

- In BPegasos with SV removal, $\| E_t \| = 1$ with Gaussian kernel. Thus, from the perspective of (6.1), all removal strategies are equivalent.
- In BNorma, the SV with the smallest norm depends on the specific choice of λ and η parameters. Therefore, the decision of which SV to remove should be made during runtime. It is worth noting that removal of the smallest SV was the strategy used by Kivinen et al. (2002) and Cheng et al. (2007) to truncate model weight for Norma.
- In BMP, $\| \boldsymbol{\alpha}_p \|^2 \, k(\mathbf{x}_p, \mathbf{x}_p) = \| \eta \prod_{i=p+1}^{t} \phi_i \|^2 \, k(\mathbf{x}_p, \mathbf{x}_p)$, because of the projection operation. Knowing that any $\phi_i \le 1$, the optimal removal will select the oldest SV. We note that removal of the oldest SV is the strategy used in Forgetron (Dekel et al., 2008).

Let us now briefly discuss other kernels, where $k(\mathbf{x}, \mathbf{x})$ in general depends on $\mathbf{x}$. In this case, the SV with the smallest norm needs to be found at runtime. How much of computational overhead this would produce depends on the particular algorithm. In case of BPegasos, this would entail finding SV with the smallest $k(\mathbf{x}_p, \mathbf{x}_p)$, while in case of Norma and BMP, it would be SV with the smallest $\| \boldsymbol{\alpha}_p \|^2 \, k(\mathbf{x}_p, \mathbf{x}_p)$ value.

6.2 Budget Maintenance through Projection

Let us consider budget maintenance through projection. In this case, before p-th SV is removed from the model, it is projected to the remaining SVs to minimize the weight degradation. By considering the multi-class case, projection can be defined as the solution of the following optimization problem,

$$\min_{\Delta\boldsymbol{\alpha}} \sum_{i\in Y} \left\| \alpha_p^{(i)}\Phi(\mathbf{x}_p) - \sum_{j\in I_{t+1}-p} \Delta\alpha_j^{(i)}\Phi(\mathbf{x}_j) \right\|^2, \tag{6.2}$$

where $\Delta\alpha_j^{(i)}$ are coefficients of the projected SV to each of the remaining SVs. After setting the gradient of (6.2) with respect to the class-specific column vector of coefficients $\Delta\boldsymbol{\alpha}^{(i)}$ to zero, one can obtain the optimal solution as

$$\forall i \in Y, \Delta\boldsymbol{\alpha}^{(i)} = \alpha_p^{(i)}\mathbf{K}_p^{-1}\mathbf{k}_p, \tag{6.3}$$

where $\mathbf{K}_p = [k_{ij}], \forall i, j \in I_{t+1} - p$ is the kernel matrix, $k_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, and $\mathbf{k}_p = [k_{pj}]^T, \forall j \in I_{t+1} - p$ is the column vector. It should be observed that inverting $\mathbf{K}_p$ can be accomplished in $O(B^2)$ time if Woodbury formula (Cauwenberghs & Poggio, 2000) is applied to reuse the results of inversion from previous projections. Finally, upon removal of the p-th SV, $\Delta\boldsymbol{\alpha}$ is added to $\boldsymbol{\alpha}$ of the remaining SVs.

The remaining issue is finding the best among B+1 candidate SVs for projection. After plugging (6.3) into (6.2) we can observe that the minimal weight degradation of projecting equals

$$\min \|\Delta_t\|^2 = \min_{p\in I_{t+1}} \|\boldsymbol{\alpha}_p\|^2 \left(k_{pp} - \mathbf{k}_p^T(\mathbf{K}_p^{-1}\mathbf{k}_p)\right). \tag{6.4}$$

Considering there are B+1 SVs, evaluation of (6.4) requires $O(B^3)$ time for each budget maintenance step. As an efficient approximation, we propose a simplified solution that always projects the smallest SV, $p = \arg\min_{j\in I_{t+1}} \|\boldsymbol{\alpha}_j\|^2\, k(\mathbf{x}_j, \mathbf{x}_j)$. Then, the computation is reduced to $O(B^2)$. We should also note that the space requirement of projection is $O(B^2)$, which is needed to store the kernel matrix and its inverse. Unlike the recently proposed projection method for multi-class perceptron (Orabona et al., 2009), that projects an SV only onto the SVs assigned to the same class, our method solves a more general case by projecting an SV onto all the remaining SVs, thus resulting in smaller weight degradation.

It should be observed that, by selecting the smallest SV to project, it can be guaranteed that weight degradation of projection is upper bounded by weight degradation of removal for any t, for all three BSGD variants from Table 1. Therefore, Theorems 1, 2, and 3 remain valid for projection. Since weight degradation for projection is expected to be, on average, smaller than that for removal, it is expected that the average error $\bar{E}$ would be smaller too, thus resulting in smaller *GapBound*.

6.3 Budget Maintenance through Merging

Problem (6.1) can also be solved by merging two SVs to a newly created one. The justification is as follows. For the i-th class weight, if $\Phi(\mathbf{x}_m)$ and $\Phi(\mathbf{x}_n)$ are replaced by $M^{(i)} = \left(\alpha_m^{(i)}\Phi(\mathbf{x}_m) + \alpha_n^{(i)}\Phi(\mathbf{x}_n)\right) / (\alpha_m^{(i)} + \alpha_n^{(i)})$ (assuming $\alpha_m^{(i)} + \alpha_n^{(i)} \neq 0$) and the coefficient of $M^{(i)}$ is set to $\alpha_m^{(i)} + \alpha_n^{(i)}$, then the weight remains unchanged. The difficulty is that $M^{(i)}$ cannot be used directly because the pre-image of $M^{(i)}$ may not exist. Moreover, even if the pre-images existed, since every class results in different $M^{(i)}$, it is not clear what M would be the best overall choice. To resolve these issues, we define the merging problem as finding an input space vector $\mathbf{z}$ whose image $\Phi(\mathbf{z})$ is at the minimum distance from the class-specific $M^{(i)}$,

$$\min_{\mathbf{z}} \sum_{i\in Y} \|M^{(i)} - \Phi(\mathbf{z})\|^2 . \tag{6.5}$$

Let us assume a radial kernel[10], $k(\mathbf{x}, \mathbf{x}') = \tilde{k}(\|\mathbf{x} - \mathbf{x}'\|^2)$, is used. Problem (6.5) can then be reduced to

[10] Gaussian kernel $k(\mathbf{x}, \mathbf{x}') = \exp(-\sigma\|\mathbf{x} - \mathbf{x}'\|^2)$ is a radial kernel.

$$\max_{\mathbf{z}} \sum_{i \in Y} (M^{(i)})^T \Phi(\mathbf{z}) . \tag{6.6}$$

Setting the gradient of (6.6) with respect to **z** to zero, leads to solution

$$\mathbf{z} = h\mathbf{x}_m + (1-h)\mathbf{x}_n, \text{ where } h = \frac{\sum_{i \in Y} m^{(i)} \tilde{k}'(\| \mathbf{x}_m - \mathbf{z} \|^2)}{\sum_{i \in Y} \left(m^{(i)} \tilde{k}'(\| \mathbf{x}_m - \mathbf{z} \|^2) + (1-m^{(i)}) \tilde{k}'(\| \mathbf{x}_n - \mathbf{z} \|^2) \right)} . \tag{6.7}$$

where $m^{(i)} = \alpha_m^{(i)} / (\alpha_m^{(i)} + \alpha_n^{(i)})$, and $\tilde{k}'(x)$ is the first derivative of $\tilde{k}$. Eq. (6.7) indicates that **z** lies on the line connecting $\mathbf{x}_m$ and $\mathbf{x}_n$. Plugging (6.7) into (6.6), the merging problem is simplified to finding

$$\max_h \sum_{i \in Y} \left(m^{(i)} k_{1-h}(\mathbf{x}_m, \mathbf{x}_n) + (1-m^{(i)}) k_h(\mathbf{x}_m, \mathbf{x}_n) \right) .$$

where we denoted $k_h(\mathbf{x}, \mathbf{x}') = k(h\mathbf{x}, h\mathbf{x}')$. We can use any efficient line search method (e.g. the golden search) to find the optimal h, which takes $O(\log(1/\varepsilon))$ time, where ε is the accuracy of the solution. After that, the optimal **z** can be calculated using (6.7).

After obtaining the optimal solution **z**, the optimal coefficient $\alpha_z^{(i)}$ for approximating $\alpha_m^{(i)}\Phi(\mathbf{x}_m) + \alpha_n^{(i)}\Phi(\mathbf{x}_n)$ by $\alpha_z^{(i)}\Phi(\mathbf{z})$ is obtained by minimizing the following objective function

$$\| \Delta_t \|^2 \equiv \min_{\alpha_z^{(i)}} \sum_{i \in Y} \left\| \alpha_m^{(i)}\Phi(\mathbf{x}_m) + \alpha_n^{(i)}\Phi(\mathbf{x}_n) - \alpha_z^{(i)}\Phi(\mathbf{z}) \right\|^2 . \tag{6.8}$$

The optimal solution of (6.8) is

$$\alpha_z^{(i)} = \alpha_m^{(i)} k(\mathbf{x}_m, \mathbf{z}) + \alpha_n^{(i)} k(\mathbf{x}_n, \mathbf{z}) .$$

The remaining question is what pair of SVs leads to the smallest weight degradation. The optimal solution can be found by evaluating merging of all $B(B-1)/2$ pairs of SVs, which would require $O(B^2)$ time. To reduce the computational cost, we use the same simplification as in projection (Section 6.2), by fixing m as the SV with the smallest value of $\| \boldsymbol{\alpha}_m \|^2$. Thus, the computation is reduced[11] to $O(B)$. We should observe that the space requirement is only $O(B)$ because there is no need to store the kernel matrix.

It should be observed that, by selecting the smallest SV to merge, it can be guaranteed that weight degradation of merging is upper bounded by weight degradation of removal for any t, for all three BSGD variants from Table 1. Therefore, Theorems 1, 2, and 3 remain valid for merging. Using the same argument as for projection, $\bar{E}$ is expected to be smaller than that of removal.

6.4 Relationship between Budget and Weight Degradation

In Theorem 1 we assumed budget maintenance is being achieved by removal. When budget maintenance is achieved by projection and merging, there is an additional impact of budget size on $\bar{E}$. As budget size B grows, the density of SVs is expected to grow. As a result, the weight degradation of projection and merging is expected to decrease. The specific amount depends on the specific data set and specific kernel. We evaluate the impact of B on $\bar{E}$ experimentally in Table 4.

7 Experiments

In this section, we evaluate BSGD[12] and compare it to related algorithms on 14 benchmark datasets.

7.1 Experimental Setting

Data sets The properties (training size, dimensionality, #classes) of 14 benchmark datasets[13] are summarized in the first row of Tables 3 (a-c). *Gauss* data was generated as a mixture of 2 two-

[11] Here, we used the fact that the kernel is radial.

[12] Our implementation is available at xxx.xxx.xxx

[13] *Adult*, *Covertype*, *DNA*, *IJCNN*, *Letter*, *Satimage*, *Shuttle* and *USPS* are available at http://www.csie.ntu.edu.tw/cjlin/libsvmtools/datasets/, *Banana* is available at

dimensional Gaussians: one class is from $N((0,0), I)$ and another is from $N((2,0), 4I)$. *Checkerboard* data was generated as a uniformly distributed two-dimensional 4×4 checkerboard with alternating class assignments. Attributes in all data sets were scaled to mean 0 and standard deviation 1.

Algorithms We evaluated several budget maintenance strategies for BSGD algorithms BPegasos, BNorma, and BMP. Specifically, we explored the following budgeted online algorithms:

- BPegasos + remove: multi-class BPegasos with arbitrary SV removal[14];
- BPegasos + project: multi-class BPegasos with projection of the smallest SV;
- BPegasos + merge: multi-class BPegasos with merging of the smallest SV;
- BNorma + merge: multi-class BNorma with merging of the smallest SV;
- BMP + merge: multi-class BPMP with merging of the smallest SV.

These algorithms were compared to the following offline, online, and budgeted online algorithms:

Offline algorithms:

- LIBSVM: state-of-art offline SVM solver (Chang and Lin, 2001); we used the “1 vs rest” method as the default setting for the multi-class tasks.

Online algorithms:

- IDSVM: online SVM algorithm which achieves the optimal solution (Cauwenberghs and Poggio, 2000);
- Pegasos: non-budgeted kernelized Pegasos (Shalev-Shwartz et al., 2011);
- Norma: non-budgeted stochastic gradient descent for kernel SVM (Kivinen et al., 2002);
- MP: non-budgeted margin perceptron algorithm (Duda and Hart, 1973) equipped with a kernel function.

Budgeted online algorithms:

- TVM: SVM-based budgeted online algorithm (Wang and Vucetic, 2010);
- BPA: budgeted Passive-Aggressive algorithm (Wang and Vucetic, 2010) that uses the projection of an SV to its nearest neighbor to maintain the budget (the BPA_{NN} version);
- MP + stop: margin perceptron algorithm that stops training when the budget is exceeded;
- MP + random: margin perceptron algorithm that removes a random SV when the budget is exceeded;
- Projectron++: margin perceptron that projects an SV only if the weight degradation is below the threshold; otherwise, budget is increased by one SV (Orabona et al., 2009). In our experiments, we set the Projectron++ threshold such that the number of SVs equals B of the budgeted algorithms at the end of training.

Gaussian kernel was used in all experiments. For Norma and BNorma, the learning rate parameter η was set either to 1 (as used by Kivinen et al., 2002) or to $0.5(2+0.5N^{-0.5})^{0.5}$ (as used by Shalev-Shwartz et al., 2011), whichever resulted in higher cross-validation accuracy. The hyper-parameters (kernel width, λ for Pegasos and Norma, U for BMP, C for LIBSVM, IDSVM, TVM) were selected by 10 fold cross-validation for each combination of data, algorithm, and budget. We repeated all the experiments five times, where at each run the training examples were shuffled differently. Mean and standard deviation of each set of experiments are reported. For *Adult*, *DNA*, *IJCNN*, *Letter*, *Pendigit*, *Satimage*, *Shuttle* and *USPS* data, we used the common training-test split. For other data sets, we randomly selected a fixed number of examples as the test data in each

---

http://ida.first.fhg.de/projects/bench/bench marks.htm, and *Waveform* data generator and *Pendigits* are available at http://archive.ics.uci.edu/ml/datasets.html.

[14] Arbitrary removal is equivalent to removing the smallest one, as discussed in Section 6.1.

repetition. We trained all online (budgeted and non-budgeted) algorithms using a single pass over the training data. All experiments were run on a 3G RAM, 3.2 GHz Pentium Dual Core. Our proposed algorithms were implemented in MATLAB.

7.2 Experimental Results

The accuracy of different algorithms on test data is reported in Table 3 (a-c)[15].

Comparison of non-budgeted algorithms. On the non-budgeted algorithm side, as expected, the exact SVM solvers LIBSVM and IDSVM have the highest accuracy[16] and are followed by Pegasos, MP and Norma, algorithms trained by a single pass of the training data. The dual-form based LIBSVM and IDSVM have sparser models than the primal-form based Pegasos, MP, and Norma. Pegasos and MP achieve similar accuracy on most data sets, while Pegasos significantly outperforms MP on the two noisy data sets *Gauss* and *Waveform*. Norma is generally less accurate than Pegasos and MP, and the gap is larger on *IJCNN*, *Checkerboard*, *DNA*, and *Covertype*. Additionally, Norma generates more SVs than its two siblings.

Comparison of budgeted algorithms. On the budgeted side, BPegasos+merge and BMP+merge are the two most accurate algorithms and their accuracies are comparable on most data sets. Considering that BPegasos+merge largely outperforms BMP+merge on *Phoneme* and *Covertype* and also the additional computational cost of the projection step in BMP, BPegasos+merge is clearly the winner of this category. The accuracy of BPegasos+project is highly competitive to the above two algorithms, but we should note that projection is costlier than merging. Accuracy of TVM and BPA is comparable to BPegasos+merge on the binary datasets (with exception of the lower BPA accuracy on IJCNN). Accuracies of Projectron++, MP+stop, and MP+random are significantly lower. In this subgroup, Projectron++ is the most successful, showing the benefits of projecting as compared to removal. Consistent with this result, BSGD algorithms using removal fared significantly worse than those using projection and merging.

[15] For IDSVM, TVM and BPA, only results on the binary datasets are reported since only binary classification versions of these algorithms are available.

Algorithms\Data		Banana (4.3K, 2, 2)	Gauss (10K, 2, 2)	Adult (21K,123,2)	IJCNN (50K, 21, 2)	Checkerb (10M, 2, 2)
Offline:						
LIBSVM	Acc:	90.70±0.06	81.62±0.40	84.29±0.0	98.72±0.10	99.87±0.02
	#SVs:	(1.1K)	(4.0K)	(8.5K)	(4.9K)	(2.6K$_{100K}$)
Online(one pass):						
IDSVM		90.65±0.04	81.67±0.40	83.93±0.03	98.51±0.03	99.40±0.02
	#SVs:	(1.1K)	(4.0K)	(4.0K$_{8.3K}$)	(3.6K$_{33K}$)	(7.5K$_{51K}$)
Pegasos		90.48±0.78	81.54±0.25	84.02±0.14	98.76±0.09	99.35±0.04
	#SVs:	(1.7K)	(6.4K)	(9K)	(16K)	(41K$_{916K}$)
Norma		90.23±1.04	81.54±0.06	83.65±0.11	93.41±0.15	99.32±0.09
	#SVs:	(2.1K)	(5.2K)	(10K)	(33K)	(128K$_{730K}$)
MP		89.40±0.57	78.45±2.18	82.61±0.61	98.61±0.10	99.43±0.11
	#SVs:	(1K)	(3.4K)	(8K)	(11K)	(22K$_{1M}$)
Budgeted online (B=100, 500):						
TVM	B = 100:	90.03±0.96	**81.56±0.16**	82.77±0.00	***97.20±0.19***	98.90±0.09
	B = 500:	***91.13±0.68***	***81.39±0.50***	***83.82±0.04***	**98.32±0.14**	**99.94±0.03**
BPA		90.35±0.37	80.75±0.24	83.38±0.56	93.01±0.53	99.01±0.04
		91.30±1.18	**81.67±0.42**	***83.58±0.30***	96.20±0.35	99.70±0.01
Projection++		88.36±1.52	76.06±2.25	77.86±3.45	92.36±1.15	96.92±0.45
		86.76±1.27	75.17±4.02	79.80±2.11	94.73±1.95	98.24±0.34
MP+stop		88.07±1.38	74.10±3.00	80.00±1.61	91.13±0.18	86.39±1.12
		89.77±0.25	79.68±1.19	81.68±0.90	94.60±0.96	95.43±0.43
MP+random		87.54±1.33	75.68±3.68	79.78±0.88	90.22±1.69	84.24±1.39
		88.36±0.99	77.26±1.16	80.40±1.03	91.86±1.39	93.12±0.56
BPegasos+remove		85.63±1.25	79.13±1.40	78.84±0.76	90.73±0.31	83.02±2.12
		89.92±0.66	80.70±0.61	81.67±0.44	93.30±0.57	91.82±0.22
BPegasos+project		***90.21±1.61***	***81.25±0.34***	83.88±0.33	96.48±0.44	97.27±0.72
		90.40±0.47	***81.33±0.40***	***83.84±0.07***	97.52±0.62	98.08±0.27
BPegasos+merge		90.17±0.61	***81.22±0.40***	**84.55±0.17**	**97.27±0.72**	**99.55±0.12**
		89.46±0.81	***81.34±0.38***	**83.93±0.41**	***98.08±0.27***	99.83±0.08
BNorma+merge		**91.53±1.14**	***81.27±0.37***	84.11±0.25	92.69±0.19	99.16±0.23
		90.65±1.28	***81.37±0.25***	***83.80±0.21***	91.35±0.13	99.72±0.05
BMP+merge		89.37±1.31	79.57±0.90	83.34±0.36	***96.67±0.35***	98.24±0.13
		89.46±0.50	79.38±0.82	82.97±0.26	***98.10±0.41***	98.79±0.08

Table 3 (a). Comparison of offline, online, and budgeted online algorithms on 5 benchmark binary classification datasets. Online algorithms (IDSVM, Pegasos, Norma and MP) were early stopped after 10,000 seconds and #examples being learned at the time of the early stopping was recorded and shown in the subscript within the #SV paranthesis. LIBSVM was trained on a subset of 100K examples on *Checkerboard*, *Covertype* and *Waveform* due to computational issues. Among the budgeted online algorithms, for each combination of data set and budget, the best mean accuracy is in bold, while the accuracies that were not significantly worse (with p > 0.05 using the one-sided t-test) are in bold and italic.

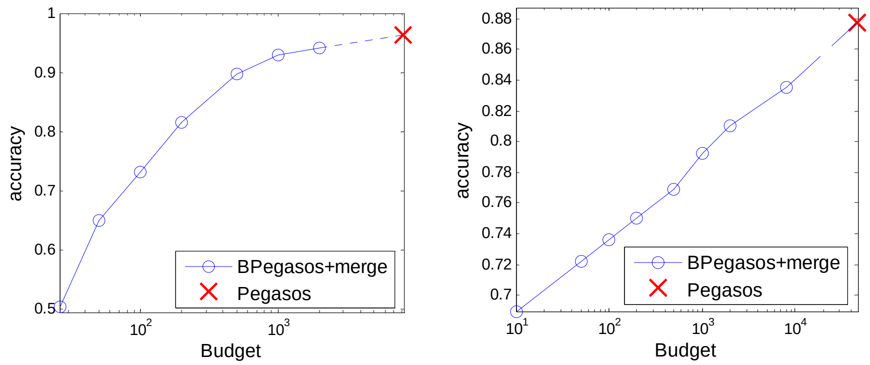
Algorithms\Data	DNA (4.3K,180, 3)	Satimage (4.4K, 36, 6)	USPS (7.3K,256,10	Pen (7.5K,16, 10)	Letter (16K, 16, 26)
Offline:					
LIBSVM	95.32±0.00	91.55±0.00	95.27±0.00	98.23±0.00	97.62±0.00
	(1.3K)	(2.5K)	(1.9K)	(0.8K)	(8.1K)
Online(one pass):					
Pegasos	92.87±0.81	91.29±0.15	94.41±0.11	97.86±0.27	96.28±0.15
#SVs:	(0.7K)	(2.9K)	(4.9K)	(1.4K)	(8.2K)
Norma	86.15±0.67	90.28±0.35	93.40±0.33	95.86±0.27	95.21±0.09
#SVs:	(2.0K)	(4.4K)	(6.6K)	(7.0K)	(15K)
MP	93.36±0.93	91.23±0.54	94.37±0.04	98.02±0.11	96.41±0.24
#SVs:	(0.8K)	(1.6K)	(2.2K)	(1.9K)	(8.2K)
Budgeted online (B=100, 500):					
Projection++	82.94±3.73	84.47±1.75	81.40±1.26	93.33±0.96	47.23±0.99
	90.11±2.11	88.66±0.66	92.02±0.59	95.78±0.75	75.90±0.76
MP+stop	73.56±7.59	82.34±2.43	79.11±2.15	88.27±1.56	41.89±1.16
	91.23±0.78	88.68±0.60	90.78±0.58	***97.78±0.20***	67.32±1.53
MP+random	73.87±4.93	82.51±1.34	78.06±2.01	87.77±2.96	40.93±2.31
	87.84±4.84	87.25±1.07	90.10±0.97	***97.20±0.68***	68.23±1.14
BPegasos+remove	78.63±2.03	81.09±3.21	80.16±1.15	91.84±1.27	41.50±1.49
	91.48±1.65	86.77±1.01	89.44±1.05	97.6±0.21	71.97±1.04
BPegasos+ project	86.53±2.03	***87.69±0.62***	89.67±0.42	96.19±0.85	74.49±1.89
	92.26±1.20	88.86±0.2	92.61±0.32	***97.58±0.49***	87.85±0.49
BPegasos+merge	***93.13±1.49***	***87.53±0.72***	***91.76±0.24***	97.06±0.19	***73.63±1.72***
	92.42±1.24	89.77±0.14	***92.91±0.19***	***97.63±0.14***	**89.68±0.61**
BNorma+merge	75.72±0.25	85.61±0.54	87.44±0.45	90.82±0.42	61.79±1.58
	76.25±3.27	86.33±0.40	89.51±0.24	94.60±0.22	75.84±0.35
BMP+merge	**93.76±0.31**	**88.33±0.90**	**92.31±0.57**	**97.35±0.16**	**74.99±1.08**
	93.84±0.64	**90.41±0.22**	**93.10±0.36**	**97.86±0.33**	88.22±0.36

Table 3 (b). Comparison of offline, online, and budgeted online algorithms on 5 benchmark multi-class datasets

Algorithms\Data	Shuttle (43K, 9, 2)	Phoneme (84K,41,48)	Covertype (0.5M, 54, 7)	Waveform (2M, 21, 3)
Offline:				
LIBSVM	99.90±0.00	78.24±0.05	89.69±0.15	85.83±0.06
	(0.3K)	(69K)	($36K_{100K}$)	($32K_{100K}$)
Online(one pass):				
Pegasos	99.90±0.00	79.62±0.16	87.73±0.31	86.50±0.10
#SVs:	(1.2K)	($80K_{80K}$)	($47K_{136K}$)	($74K_{192K}$)
Norma	99.79±0.01	79.86±0.09	82.80±0.33	86.29±0.15
#SVs:	(8K)	(84K)	($92K_{92K}$)	($111K_{189K}$)
MP	99.89±0.02	79.80±0.12	88.84±0.06	84.36±0.36
#SVs:	($0.4K_{44K}$)	($78K_{78K}$)	($56K_{160K}$)	($83K_{310K}$)
Budgeted online (B=100, 500):				
Projection++	99.55±0.16	21.20±1.24	62.54±3.14	80.75±0.81
	99.85±0.08	32.32±1.97	67.32±2.93	83.56±0.54
MP+stop	99.39±0.35	24.86±2.10	56.96±1.59	81.04±2.61
	99.90±0.01	33.76±1.01	61.93±1.56	83.76±0.71
MP+random	98.67±0.07	23.29±1.39	55.56±1.37	79.94±1.12
	99.90±0.01	31.37±1.91	60.47±1.70	81.61±1.51
BPegasos+remove	99.26±0.54	24.39±1.48	55.64±1.82	78.43±1.79
	99.89±0.02	32.10±0.85	62.97±0.55	84.38±0.53
BPegasos+ project	**99.81±0.05**	43.60±0.10	70.84±0.59	85.63±0.07
	99.89±0.02	48.87±0.07	74.94±0.22	86.18±0.06
BPegasos+merge	99.63±0.02	**46.49±0.78**	**74.10±0.30**	**86.71±0.38**
	99.89±0.02	**51.57±0.30**	**76.89±0.51**	**86.63±0.28**
BNorma+merge	99.48±0.01	39.66±0.66	71.54±0.53	***86.60±0.12***
	99.80±0.01	45.13±0.43	72.81±0.46	82.03±0.53
BMP+merge	98.99±0.55	42.18±1.94	67.28±3.86	86.02±0.22
	99.91±0.01	47.02±0.98	72.31±0.75	86.03±0.17

Table 3 (c). Comparison of offline, online, and budgeted online algorithms on 4 benchmark multi-class datasets

Best budgeted algorithm vs non-budgeted algorithms. Comparing the best budgeted algorithm BPegasos+merge with modest budgets of B = 100 and 500 with the non-budgeted Pegasos and LIBSVM, we can see that it achieves very competitive accuracy. Interestingly, its accuracy is even larger than the two non-budgeted algorithms on two largest data sets *Checkerboard* and *Waveform*. This indicates noise reduction capability of SV merging. This result is even more significant as BPegasos+merge has faster training time and learns a much smaller model. On *Covertype, Phoneme* and *Letter* data, the accuracy gap between budget B = 500 and non-budgeted algorithms remained large and it can be explained by the complexity of these problems; for example, 30% of *Covertype* examples, 50% of *Letter* examples, and 100% of *Phoneme* examples became SVs in Pegasos. In addition, *Letter* had 26 class labels and Phoneme 48. In all 3 data sets, the accuracy clearly improved from B = 100 to 500, which indicates that extra budget is needed for comparable accuracy. To better illustrate the importance of budget size on some data sets, Figure 2 shows that on *Letter* and *Covertype*, the accuracy of BPegasos+merge approaches that of Pegasos as the larger budget is used. Interestingly, while 16K examples appear to be sufficient for convergence on *Letter* data set, it is evident that *Covertype* could benefit from a much larger budget than the available one with its half million labeled examples.



(a) Letter (b) Covertype

Figure 2. The accuracy of BPegasos as a function of the budget size

Multi-epoch vs single-pass training. For the most accurate budgeted online algorithm BPegasos+merge, we also report its accuracy after allowing it to make 5 passes through training data. In this scenario, BPegasos should be treated as an offline algorithm. The accuracy improvement as compared to the single-pass version is reported in Figure 3 (data sets are indexed following the order in Tables 3.a-c). We can observe that multi-epoch training improves the accuracy of BPegasos on most data sets. This result suggests that, if the training time is not of major concern, multiple accesses to the training data should be used.

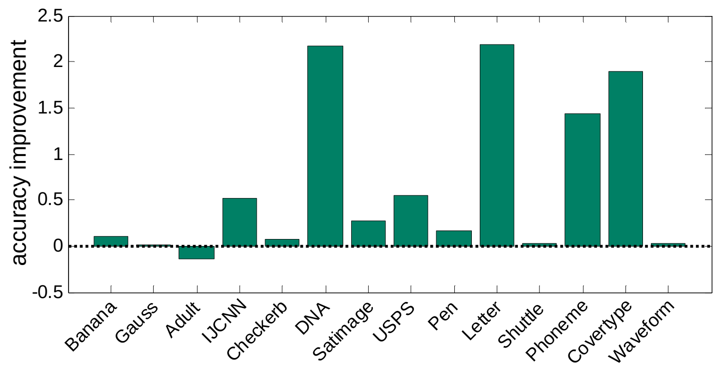
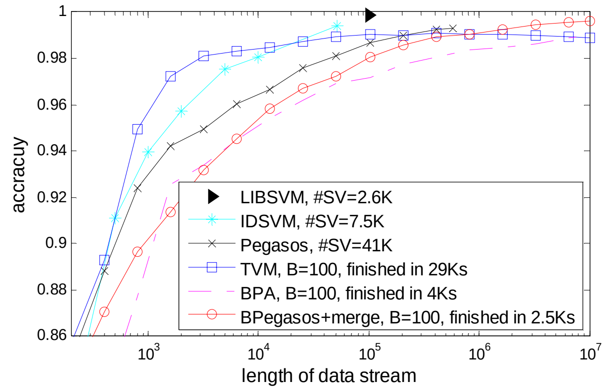
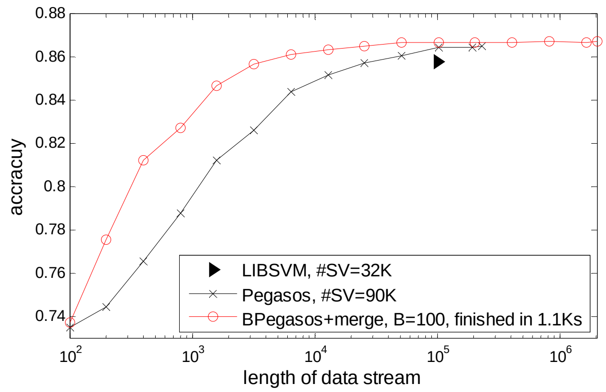


Figure 3. BPegasos+merge (B=500): accuracy improvement of 5-passes vs single-pass training.

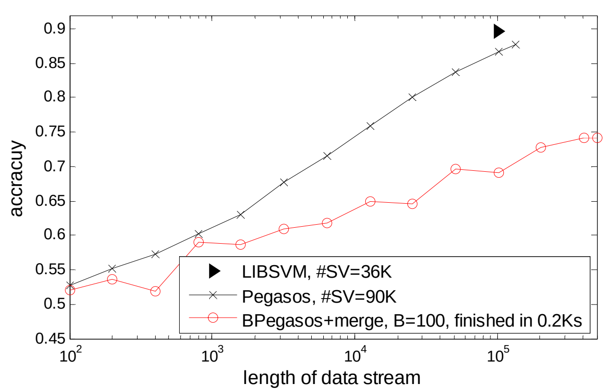
Accuracy evolution curves. In Figure 4 we show evolution of the accuracy as a function of number of observed examples on the three largest data sets. By comparing BPegasos+merge with non-budgeted SVMs and several other budgeted algorithms from Table 3, we observe that the accuracy of BPegasos+merge consistently increases with data stream size. On *Checkerboard*, its accuracy closely follows Pegasos and eventually surpasses it after Pegasos had to be early stopped. IDSVM and its budgeted version TVM exhibit faster accuracy growth initially, but are surpassed by BPegasos+merge as more training examples become available. On waveform, the accuracy of budgeted Pegasos grows faster than the original non-budgeted version. This behavior can be attributed to the noise-reduction property of merging. Finally, on *Covertype*, BPegasos+merge significantly trails its non-budgeted cousin, and this behavior is consistent with Figure 2.b.



(a) Checkerboard

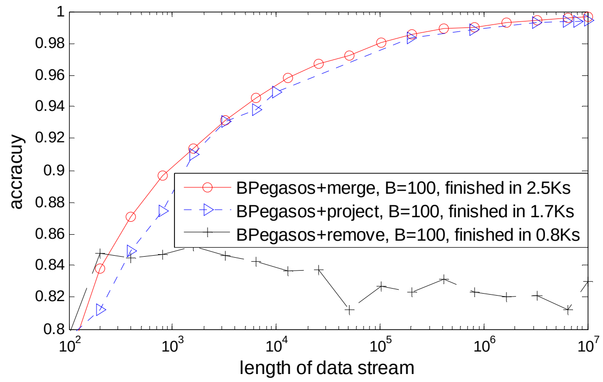


(b) Waveform

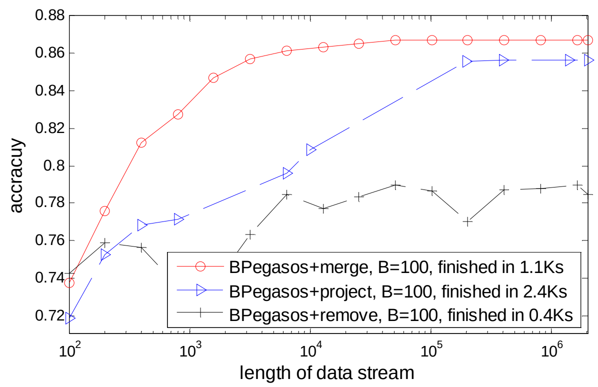


(c) Covertype

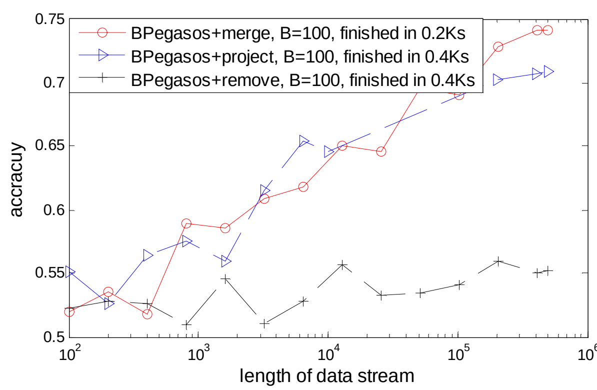
Figure 4. Comparison of accuracy evolution curves



(a) Checkerboard



(b) Waveform



(c) Covertype

Figure 5. Accuracy evolution curves of BPegasos for different budget maintenance strategies

In Figure 5 we compare evolution of accuracy of BPegasos for 3 proposed budget maintenance strategies. As could be seen, removal is inferior to projection and merging on all 3 data sets. On *Checkerboard*, removal even causes a gradual drop in accuracy after a initial moderate increase, while on the other two sets the accuracy fluctuates around a very small value close to that achieved by training on 100 initial examples. On the other hand, projection and merging result in strong and consistent accuracy increase with data stream size. Interestingly, on *Waveform* data, merging significantly outperforms projection, which may point to its robustness to noise.

Both Figures 4 and 5 list total training time of budgeted algorithms at the end of the data stream (non-budgeted algorithms were early stopped after 10K seconds). Considering that our implementation of all algorithms except LIBSVM was in Matlab and on a 3GB RAM, 3.2GHz

Pentium Dual Core 2 PC, Figure 4 indicates a rather impressive speed of the budgeted algorithms. From Figure 5, it can be seen that merging and projection are very fast and are comparable to removal.

Training time scalability. Figure 6 presents log-log plot of the training time versus the data stream length on *Checkerboard* data set with 10 million examples. Excluding the initial stage, Pegasos had the fastest increase in training time, confirming the expected $O(N^2)$ runtime. On the budgeted side, the runtime time of BPegasos with merging and projecting increases linearly with data size. However, it is evident that costs of projecting grow much faster than costs of merging with the budget size. This confirms the expected $O(B)$ scaling of merging and $O(B^2)$ scaling of projection.

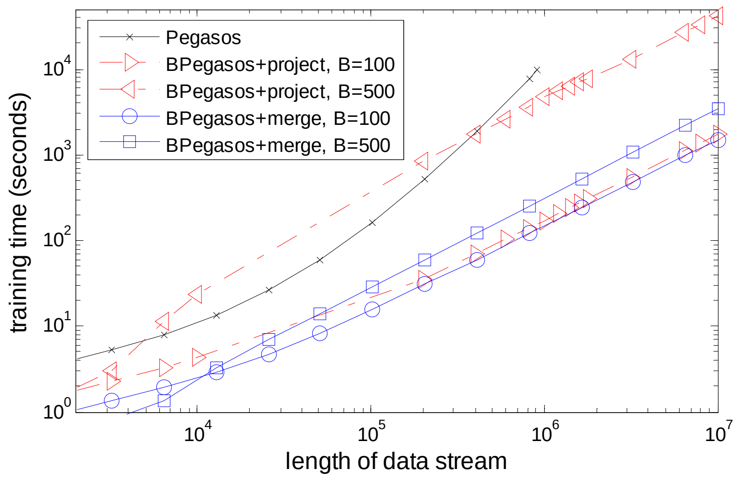


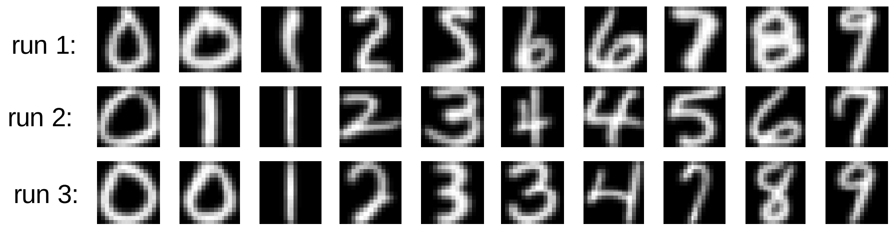
Figure 6. Training time curves on *Checkerboard* data

Weight degradation. Theorems 1, 2 and 3 indicate that lower $\bar{E}$ leads to lower gap between the optimal solution and the budgeted solution on *Checkerboard* data. We also argued that $\bar{E}$ decreases with budget size through three mechanisms. In Table 4, we show how the value of $\bar{E}$ is being influenced by the budget B and, in turn, how the change in $\bar{E}$ influences the accuracy. From the comparison of three strategies for two B values (100 and 500), we do see that smaller $\bar{E}$ leads to higher accuracy. For each specific strategy, as B gets larger, $\bar{E}$ is getting smaller. Those results verify that projection and merging achieve significantly lower $\bar{E}$ value than removal and that lower $\bar{E}$ indeed results in higher accuracy.

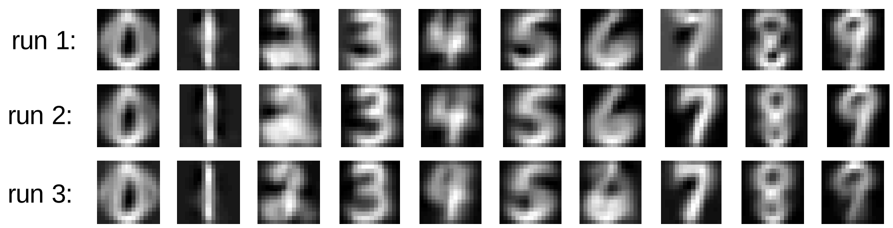
	B = 100		B = 500	
	Acc	$\bar{E}$	Acc	$\bar{E}$
BPegasos+remove	79.19±3.05	1.402±0.000	90.32±0.40	1.401±0.000
BPegasos+project	99.25±0.06	0.052±0.007	99.66±0.10	0.007±0.001
BPegasos+merge	99.55±0.14	0.037±0.006	99.74±0.08	0.002±0.001

Table 4. Comparison of accuracy and averaged model degradation for three versions of BPegasos as a function of budget size B on 10M Checkerboard examples, using same parameters ($\lambda = 10^{-4}$, kernel width $\sigma = 0.0625$).

Merged vs projected SVs. In order to gain further insights into the projection and merging-based budget maintenance, in Figure 7 we compare final SVs generated by these two strategies on the *USPS* data where classes are 10 digits. We used budget B = 10 for BPegasos to explore how successful the algorithm was in revealing the 10 digits. Comparing Figures 7.a and 7.b we can observe that SVs generated by 3 different runs of BPegasos+project did not represent all 10 digits (e.g. in Run 1, digits 0 and 6 appear twice, while digits 3 and 4 are not represented). It should be noted that the 10 SVs obtained using projection are identical to actual 10 training examples of USPS. On the other hand, SVs obtained by merging in all 3 runs represent all 10 digits. The appearance of each SV in Figure 7.b is blurred and is a result of many mergings of the original labeled examples. This example is a useful illustration of the main difference between projection and merging, and it can be helpful in selecting the appropriate budget maintenance strategy for a particular learning task.



(a) 3 different runs using BPegasos+project (B = 10) with average accuracy 74%



(b) 3 different runs using BPegasos+merge (B = 10) with average accuracy 86%

Figure 7. The plot of SVs on *USPS* data. (each row corresponds to a different run)

8 Conclusion

We proposed a framework for large-scale kernel SVM training using Budgeted Stochastic Gradient Descent (BSGD) algorithms. We showed that budgeted versions of three popular online learning algorithms, Pegasos, Norma, and Margin Perceptron, can be studied under this framework. We obtained theoretical guarantees for their performance that indicate that BSGD accuracy is closely related to the model degradation due to budget maintenance. Based on the analysis, we studied budget maintenance strategies based on removal, projection, and merging. We experimentally evaluated the proposed BSGD algorithms in terms of accuracy and training time and compared them with a number of offline and online, non-budgeted, and budgeted alternatives. The results indicate that highly accurate and compact kernel SVM classifiers can be trained on high-throughput data streams. Particularly, the results show that merging is a highly attractive budget maintenance strategy for BSGD algorithms as it results in relative accurate classifiers while achieving linear training time scaling with support vector budget and data size.

Acknowledgement

This work was supported by the U.S. National Science Foundation Grant IIS-0546155. Koby Crammer is a Horev Fellow, supported by the Taub Foundations.

Appendix

A. Proof of Theorem 1

We start by showing the following technical lemma.

Lemma 1

- *Let P_t be as defined in (3.3).*

- *Let C be a closed convex set with radius U.*
- *Let* $\mathbf{w}_1,\ldots,\mathbf{w}_N$ *be a sequence of vectors such that* $\mathbf{w}_1 \in C$ *and for any* $t > 1$ $\mathbf{w}_{t+1} \leftarrow \prod_C(\mathbf{w}_t - \eta_t \nabla_t - \Delta_t)$, *where* ∇_t *is the (sub)gradient of* P_t *at* $\mathbf{w}_t$, η_t *is a learning rate function,* Δ_t *is a vector, and* $\prod_C(\mathbf{w}) = \arg\min_{\mathbf{w}' \in C} \|\mathbf{w}' - \mathbf{w}\|$ *is a projection operation that projects* $\mathbf{w}$ *to C.*
- *Assume* $\| E_t \| \le 1$.
- *Define* $D_t = \|\mathbf{w}_t - \mathbf{u}\|^2 - \|\mathbf{w}_{t+1} - \mathbf{u}\|^2$ as the relative progress toward $\mathbf{u}$ at t-th round.

Then, the following inequality holds for any $\mathbf{u} \in C$,

$$\frac{1}{N}\sum_{t=1}^{N} P_t(\mathbf{w}_t) - \frac{1}{N}\sum_{t=1}^{N} P_t(\mathbf{u}) \le \frac{1}{N}\left(\sum_{t=1}^{N}\frac{D_t}{2\eta_t} - \sum_{t=1}^{N}\frac{\lambda}{2}\|\mathbf{w}_t - \mathbf{u}\|^2 + \frac{(\lambda U + 2)^2}{2}\sum_{t=1}^{N}\eta_t\right) + 2U\overline{E} \qquad (A.1)$$

Proof of Lemma 1:

First, we rewrite $\mathbf{w}_{t+1} \leftarrow \prod_C(\mathbf{w}_t - \eta_t \nabla_t - \Delta_t)$ by treating Δ_t as the source of error in the gradient $\mathbf{w}_{t+1} \leftarrow \prod_C(\mathbf{w}_t - \eta_t \partial_t)$, where we defined $\partial_t = \nabla_t + E_t$.

Then, we lower bound D_t as

$$\begin{aligned} D_t &= \|\mathbf{w}_t - \mathbf{u}\|^2 - \|\textstyle\prod_C(\mathbf{w}_t - \eta_t \partial_t) - \mathbf{u}\|^2 \\ &\ge_1 \|\mathbf{w}_t - \mathbf{u}\|^2 - \|\mathbf{w}_t - \eta_t \partial_t - \mathbf{u}\|^2 \\ &= -\eta_t^2 \|\partial_t\|^2 + 2\eta_t \nabla_t^T(\mathbf{w}_t - \mathbf{u}) + 2\eta_t E_t^T(\mathbf{w}_t - \mathbf{u}) \\ &\ge_2 -\eta_t^2(\lambda U + 1 + 1)^2 + 2\eta_t\left(P_t(\mathbf{w}_t) - P_t(\mathbf{u}) + \frac{\lambda}{2}\|\mathbf{w}_t - \mathbf{u}\|^2\right) - 4\eta_t \|E_t\| U. \end{aligned} \qquad (A.2)$$

In $\ge_1$, we use the fact that since C is convex, $\|\prod_C(a) - b\| \le \|a - b\|$ for all $b \in C$ and a. In $\ge_2$, $\|\partial_t\|$ is bounded as

$$\|\partial_t\| \le \|\lambda \mathbf{w}_t + y_t \Phi(\mathbf{x}_t)\| + \|E_t\| \le \lambda U + 1 + 1,$$

and, by applying the property of strong convexity, it follows

$$\nabla_t^T(\mathbf{w}_t - \mathbf{u}) \ge P_t(\mathbf{w}_t) - P_t(\mathbf{u}) + \lambda\|\mathbf{w}_t - \mathbf{u}\|^2/2,$$

since P_t is λ-strongly convex function w.r.t. $\|\mathbf{w}\|^2/2$ and ∇_t is the subgradient of $P_t(\mathbf{w})$ at $\mathbf{w}_t$ (according to Lemma 1 by Shalev-Shwartz & Singer, 2007). Bound $\|\mathbf{w}_t - \mathbf{u}\| \le 2U$ holds since both $\|\mathbf{w}_t\|$ and $\|\mathbf{u}\|$ are upper bounded by U.

Dividing both sides of inequality (A2) by $2\eta_t$ and rearranging, we obtain

$$P_t(\mathbf{w}_t) - P_t(\mathbf{u}) \le \frac{D_t}{2\eta_t} - \frac{\lambda}{2}\|\mathbf{w}_t - \mathbf{u}\|^2 + \frac{\eta_t(\lambda U + 1 + 1)^2}{2} + 2\|E_t\|U. \qquad (A.3)$$

Summing over all t in (A.3) and dividing two sides of inequality by N leads to the stated bound.

■

Proof of Theorem 1:

$\mathbf{w}_{t+1}$ is bounded as

$$\begin{aligned} \|\mathbf{w}_{t+1}\| &= \|(1 - \eta_t\lambda)\mathbf{w}_t + \eta_t y_t \Phi(\mathbf{x}_t) - \Delta_t\| \\ &\le \|(1 - \eta_t\lambda)\mathbf{w}_t\| + \eta_t(1 + \|E_t\|) \le_3 2/\lambda. \end{aligned}$$

In $\le_3$ we used the definition of η_t and recursively bounded $\|\mathbf{w}_t\|$.

Using the fact that $\|\mathbf{w}^*\| \le 1/\sqrt{\lambda}$ (Shalev-Shwartz, Singer & Srebro, 2011), both $\|\mathbf{w}_t\|$ and $\|\mathbf{w}^*\|$ can be bounded by constant U defined in (4.1). Thus the update rule $\mathbf{w}_{t+1} \leftarrow \prod_C(\mathbf{w}_t - \eta_t \nabla_t - \Delta_t)$ in Lemma 1 is reduced to $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla_t - \Delta_t$.

Plugging $\eta_t \equiv 1/(\lambda t)$ into RHS of inequality (A.1) in Lemma 1 and replacing $\mathbf{u}$ with $\mathbf{w}^*$, the first and second term in the parenthesis on the RHS are bounded as

$$\sum_{t=1}^{N}\frac{D_t}{2\eta_t}-\sum_{t=1}^{N}\frac{\lambda}{2}\|\mathbf{w}_t-\mathbf{w}^*\|^2$$
$$\le\frac{1}{2}\left((\frac{1}{\eta_1}-\lambda)\|\mathbf{w}_1-\mathbf{w}^*\|^2+\sum_{t=2}^{N}(\frac{1}{\eta_t}-\frac{1}{\eta_{t-1}}-\lambda)\|\mathbf{w}_t-\mathbf{w}^*\|^2-\frac{1}{\eta_N}\|\mathbf{w}_{N+1}-\mathbf{w}^*\|^2\right)$$
$$=-\frac{1}{2\eta_N}\|\mathbf{w}_{N+1}-\mathbf{w}^*\|^2\le 0. \tag{A.4}$$

According to the divergence rate of harmonic series, the third term on the RHS in (A.1) is bounded by,

$$\frac{(\lambda U+2)^2}{2}\sum_{t=1}^{N}\eta_t=\frac{(\lambda U+2)^2}{2\lambda}\sum_{t=1}^{N}\frac{1}{t}\le\frac{(\lambda U+2)^2}{2\lambda}(\ln(N)+1)\,. \tag{A.5}$$

With bounded U value, combining (A.4), (A.5) with (A.1) leads to (4.2). ■

B. Proof of Theorem 2

$\mathbf{w}_{t+1}$ is bounded as

$$\begin{aligned}\|\mathbf{w}_{t+1}\|&=\|(1-\eta_t\lambda)\mathbf{w}_t+\eta_t y_t\Phi(\mathbf{x}_t)-\Delta_t\|\\&\le\|(1-\eta_t\lambda)\mathbf{w}_t\|+\eta_t(1+\|E_t\|)\\&\le\|(1-\eta_t\lambda)\mathbf{w}_t\|+2\eta_t.\end{aligned}$$

Since $\|\mathbf{w}_1\|=0$, the bound $\|\mathbf{w}_{t+1}\|\le 2/\lambda$ holds for all t (Kivinen et al., 2002). Using the fact that $\|\mathbf{w}^*\|\le 1/\sqrt{\lambda}$ (Shalev-Shwartz, Singer & Srebro, 2011), both $\|\mathbf{w}_t\|$ and $\|\mathbf{w}^*\|$ are bounded by constant U defined in (4.1). Thus the update rule $\mathbf{w}_{t+1}\leftarrow\prod_C(\mathbf{w}_t-\eta_t\nabla_t-\Delta_t)$ in Lemma 1 is reduced to $\mathbf{w}_{t+1}\leftarrow\mathbf{w}_t-\eta_t\nabla_t-\Delta_t$.

Replacing **u** by $\mathbf{w}^*$ in (A.1), the first term at RHS in (A.1) can be bounded as

$$\sum_{t=1}^{N}\frac{D_t}{2\eta_t}=\sum_{t=1}^{N}\frac{1}{2\eta_t}(\|\mathbf{w}_t-\mathbf{w}^*\|^2-\|\mathbf{w}_{t+1}-\mathbf{w}^*\|^2)$$
$$=\frac{1}{2\eta_1}\|\mathbf{w}_1-\mathbf{w}^*\|^2+\frac{1}{2}\sum_{t=2}^{N}(\frac{1}{\eta_t}-\frac{1}{\eta_{t-1}})\|\mathbf{w}_t-\mathbf{w}^*\|^2-\frac{1}{2\eta_N}\|\mathbf{w}_{N+1}-\mathbf{w}^*\|^2$$
$$\le\frac{4U^2}{2}\left(\frac{1}{\eta_1}+\sum_{t=2}^{N}(\frac{1}{\eta_t}-\frac{1}{\eta_{t-1}})\right)=\frac{2U^2}{\eta_N}=\frac{2U^2\sqrt{N}}{\eta} \tag{B.1}$$

The third term at RHS in (A.1) can be bounded according to the series divergence rate as

$$\frac{(\lambda U+2)^2}{2}\sum\nolimits_{t=1}^{N}\eta_t\le\frac{\eta(\lambda U+2)^2}{2}(2\sqrt{N}-1)\,. \tag{B.2}$$

With bounded U value, combing (B.1), (B.2) with (A.1) and bounding the negative terms by zero lead to (4.3). ■

C. Proof of Theorem 3

Replacing **u** by $\mathbf{w}^*$ in (A.1), the first term at RHS in (A.1) can be bounded as

$$\sum_{t=1}^{N}\frac{D_t}{2\eta_N}=\sum_{t=1}^{N}\frac{1}{2\eta_t}\left(\|\mathbf{w}_t-\mathbf{w}^*\|^2-\|\mathbf{w}_{t+1}-\mathbf{w}^*\|^2\right)$$
$$=\frac{1}{2\eta_1}\|\mathbf{w}_1-\mathbf{w}^*\|^2+\frac{1}{2}\sum_{t=2}^{N}(\frac{1}{\eta_t}-\frac{1}{\eta_{t-1}})\|\mathbf{w}_t-\mathbf{w}^*\|^2-\frac{1}{2\eta_N}\|\mathbf{w}_{N+1}-\mathbf{w}^*\|^2$$
$$=\frac{1}{2\eta}\|\mathbf{w}_1-\mathbf{w}^*\|^2-\frac{1}{2\eta}\|\mathbf{w}_{N+1}-\mathbf{w}^*\|^2\le\frac{U^2}{2\eta}$$

using the fact $\eta_t = \eta$ and $\lambda = 0$.

Bounding the second term in RHS of (A.1) by zero (since it is always negative), Lemma 1 directly leads to (4.4). ■

References

A. Bordes, L. Bottou and P. Gallinari, (2009). SGD-QN: careful quasi-newton stochastic gradient descent. *Journal of Machine Learning Research*.

A. Bordes, S. Ertekin, J. Weston, and L. Bottou, (2005). Fast kernel classifiers for online and active learning. *Journal of Machine Learning Research*.

C. J. C. Burges, (1996). Simplified support vector decision rules. *ICML*.

G. Cauwenberghs and T. Poggio, (2000). Incremental and decremental support vector machine learning. *NIPS*.

N. Cesa-Bianchi and C. Gentile, (2006). Tracking the best hyperplane with a simple budget Perceptron. *COLT*.

C.-C. Chang and C.-J. Lin, (2001). LIBSVM: a library for support vector machines, http://www.csie.ntu.edu.tw/~cjlin/libsvm.

F. Chang, C.-Y. Guo, X.-R. Lin and C.-J. Lu. (2010). Tree Decomposition for Large-Scale SVM Problems. *Journal of Machine Learning Research*.

Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard and C.-J. Lin, (2010). Training and testing low-degree polynomial data mappings via linear svm. *Journal of Machine Learning Research*.

L. Cheng, S. V. N. Vishwanathan, D. Schuurmans, S. Wang and T. Caelli. (2007). Implicit online earning with kernels. *NIPS*.

R. Collobert, F. Sinz, J. Weston and L. Bottou, (2006). Trading convexity for scalability. *ICML*.

C. Cortes, & V. Vapnik, (1995). Support-vector networks. *Machine Learning*.

K. Crammer, J. Kandola and Y. Singer, (2004). Online classification on a budget. *NIPS*.

K. Crammer and Y. Singer, (2001). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*.

K. Crammer and Y. Singer. (2003). Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*.

K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz and Y. Singer (2006). Online passive-aggressive algorithms. *Journal of Machine Learning Research*.

L. Csató and M. Opper, (2001). Sparse representation for gaussian process models. NIPS.

O. Dekel, S. Shalev-Shwartz and Y. Singer, (2008). The forgetron: a kernel-based perceptron on a budget. *SIAM Journal on Computing*.

O. Dekel and Y. Singer, (2006). Support vector machines on a budget. *NIPS*.

R. Duda and P. Hart, (1973). Pattern classification and scene analysis. New York: Wiley.

Y. Engel, S. Mannor and R. Meir, (2002). Sparse online greedy support vector regression. *ECML*.

C. Gentile, (2001). A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*.

H.-P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. (2005). Parallel support vector machines: the cascade svm. *NIPS*.

C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan, (2008). A dual coordinate descent method for large-scale linear svm. *ICML*.

C.-W. Hsu and C.-J. Lin, (2002). A comparisons of methods multiclass support vector machines. *IEEE Transactions on Neural Networks*.

T. Joachims, (2006). Training linear svms in linear time. *KDD*.

S. S. Keerthi, O. Chapelle, and D. DeCoste, (2006). Building support vector machines with reduced classifier complexity. *Journal of Machine Learning Research*.

J. Kivinen, A. J. Smola, and R. C. Williamson, (2002). Online learning with kernels. *IEEE Transactions on Signal Processing*.

Y.-J. Lee and O. L. Mangasarian, (2001). RSVM: reduced support vector machines. *SDM*.

B. Li, M. Chi, J. Fan, and X. Xue, (2007). Support cluster machine. *ICML*.

Y. Li and P. Long, (2002). The relaxed online maximum margin algorithm. *Machine Learning*.

D. Nguyen and T. Ho, (2005). An efficient method for simplifying support vector machines. *ICML*.

F. Orabona, J. Keshet and B. Caputo, (2009). Bounded kernel-based online learning. *Journal of Machine Learning Research*.

J. C. Platt, (1998). Fast training of support vector machines using sequential minimal optimization. Advances in kernel methods - support vector learning, *MIT Press*.

A. Rahimi and B. Recht, (2007). Random features for large-scale kernel machines. *NIPS*.

F. Rosenblatt, (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*.

B. Schökopf, S. Mika, C. J. C. Burges, P. Knirsch, K. Müler, G. Räsch and A. J. Smola, (1999). Input space versus feature space in kernel-based methods. *IEEE Transactions on Neural Networks*.

S. Shalev-Shwartz and Y. Singer, (2007). Logarithmic regret algorithms for strongly convex repeated games (Technical Report). The Hebrew University.

S. Shalev-Shwartz, Y. Singer, N. Srebro and A. Cotter (2011). Pegasos: primal estimated subgradient solver for svm. *Mathematical Programming*.

S. Sonnenburg and V. Franc, (2010), COFFIN: a computational framework for linear svms. *ICML*.

I. Steinwart, (2003). Sparseness of support vector machines. *Journal of Machine Learning Research*.

I. Sutskever, (2009). A simpler unified analysis of budget perceptrons. *ICML*.

C.H. Teo, S. V. N. Vishwanathan, A. J. Smola and Q. V. Le, (2010). Bundle Methods for Regularized Risk Minimization. *Journal of Machine Learning Research*.

S. V. N. Vishwanathan, A. J. Smola, and M. N.Murty, (2003). SimpleSVM. *ICML*.

I. W. Tsang, J. T. Kwok, and P.-M. Cheung, (2005). Core vector machines: fast svm training on very large data sets. *Journal of Machine Learning Research*.

Z. Wang and S. Vucetic, (2009). Tighter perceptron with improved dual use of cached data for model representation and validation. *IJCNN*.

Z. Wang and S. Vucetic, (2010a). Online training on a budget of support vector machines using twin prototypes. *Statistical Analysis and Data Mining*.

Z. Wang and S. Vucetic, (2010b). Online passive-aggressive algorithms on a Budget. *AISTATS*.

Z. Wang, N. Djuric, K. Crammer and S. Vucetic, (2011). Trading representability for scalability: adaptive multi-hyperplane machine for nonlinear classification. *KDD*.

J. Weston, A. Bordes and L. Bottou, (2005). Online (and offline) on an even tighter budget. *AISTATS*.

M.-R. Wu, B. Schökopf, and G. Barik, (2005). Building sparse large margin classifiers. *ICML*.

H.-F. Yu, C.-J. Hsieh, K.-W. Chang, and C.-J. Lin (2010). Large linear classification when data cannot fit in memory. *KDD*.

M. Zinkevich, (2003). Online convex programming and generalized infinitesimal gradient ascent. *ICML*.

K. Zhang, L. Lan, Z. Wang and F. Moerchan, (2012). Scaling up Kernel SVM on Limited Resources: a Low-rank Linearization Approach. *AISTATS*.

T. Zhang, (2004). Solving large scale linear prediction problems using stochastic gradient descent. *ICML*.

Z. A. Zhu, W. Chen, G. Wang, C. Zhu and Z. Chen, (2009). P-packSVM: parallel primal gradient descent kernel svm. *ICDM*.