Online Passive-Aggressive Algorithms on a Budget

Zhuang Wang
Dept. of Computer and Information Sciences
Temple University, USA
zhuang@temple.edu

Slobodan Vucetic
Dept. of Computer and Information Sciences
Temple University, USA
vucetic@temple.edu

Abstract

In this paper a kernel-based online learning algorithm, which has both constant space and update time, is proposed. The approach is based on the popular online Passive-Aggressive (PA) algorithm. When used in conjunction with kernel function, the number of support vectors in PA grows without bounds when learning from noisy data streams. This implies unlimited memory and ever increasing model update and prediction time. To address this issue, the proposed budgeted PA algorithm maintains only a fixed number of support vectors. By introducing an additional constraint to the original PA optimization problem, a closed-form solution was derived for the support vector removal and model update. Using the hinge loss we developed several budgeted PA algorithms that can trade between accuracy and update cost. We also developed the ramp loss versions of both original and budgeted PA and showed that the resulting algorithms can be interpreted as the combination of active learning and hinge loss PA. All proposed algorithms were comprehensively tested on 7 benchmark data sets. The experiments showed that they are superior to the existing budgeted online algorithms. Even with modest budgets, the budgeted PA achieved very competitive accuracies to the non-budgeted PA and kernel perceptron algorithms.

Appearing in Proceedings of the 13^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy. Volume 9 of JMLR: W&CP 9.

1 INTRODUCTION

The introduction of Support Vector Machines (Cortes and Vapnik, 1995) generated significant interest in applying the kernel methods for online learning. A large number of online algorithms (e.g. perceptron (Rosenblatt, 1958) and PA (Crammer et al., 2006)) can be easily kernelized. The online kernel algorithms have been popular because they are simple, can learn nonlinear mappings, and could achieve state-of-the-art accuracies. Perhaps surprisingly, online kernel classifiers can place a heavy burden on computational resources. The main reason is that the number of support vectors that need to be stored as part of the prediction model grows without limit as the algorithm progresses. In addition to the potential of exceeding the physical memory, this property also implies an unlimited growth in model update and perdition time.

Budgeted online kernel classifiers have been proposed to address the problem of unbounded growth in computational cost through maintaining a limited number of support vectors during training. The originally proposed budgeted algorithm (Crammer et al., 2004) achieves this by removing a support vector every time the budget is exceeded. The algorithm has constant space and prediction time. Support vector removal is an underlying idea of several other budgeted algorithms. These include removal of a randomly selected support vector (Cesa-Bianchi and Gentile, 2007; Vucetic et al. 2009), the oldest support vector (Dekel et al., 2008), the support vector with the smallest coefficient (Cheng et al., 2007) and the support vector resulting in the smallest error on the validation data (Weston et al., 2005). Recently studied alternatives to removal includes projecting an SV prior to its removal (Orabona et al., 2008) and merging two SVs into a new one (Wang and Vucetic, 2010). In practice, the assigned budget depends on the specific application requirements (e.g. memory limitations, processing speed, data throughput).

In this paper, we propose a family of budgeted on-

line classifiers based on the online Passive-Aggressive algorithm (PA). PA (Crammer, et al., 2006) is a popular kernel-based online algorithm that has solid theoretical guarantees, is as fast as and more accurate than kernel perceptrons. Upon receiving a new example, PA updates the model such that it is similar to the current one and has a low loss on the new example. When equipped with kernel, PA suffers from the same computational problems as other kernelized online algorithms, which limits its applicability to large-scale learning problems or in the memory-constrained environments. In the proposed budgeted PA algorithms, the model updating and budget maintenance are solved as a joint optimization problem. By adding an additional constraint into the PA optimization problem the algorithm becomes able to explicitly bound the number of support vectors by removing one of them and by properly updating the rest. The new optimization problem has a closed-from solution. Under the same underlying algorithmic structure, we develop several algorithms with different tradeoffs between accuracy and computational cost.

We also propose to replace the standard hinge loss with the recently proposed ramp loss in the optimization problem such that the resulting algorithms are more robust to the noisy data. We use an iterative method, ConCave Convex Procedure (CCCP) (Yuille and Rangarajan, 2002), to solve the resulting non-convex problem. We show that CCCP converges after a single iteration and results in a closed-from solution which is identical to its hinge loss counterpart when it is coupled with active learning.

We evaluated the proposed algorithms against the state-of-the-art budgeted online algorithms on a large number of benchmark data sets with different problem complexity, dimensionality and noise levels.

2 PROBLEM SETTING AND BACKGROUND

We consider online learning for binary classification, where we are given a data stream D consisting of examples $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$, where $\mathbf{x} \in \mathbb{R}$ is an M-dimensional attribute vector and $y \in \{+1, -1\}$ is the associated binary label. PA is based on the linear prediction model of the form $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, where $\mathbf{w} \in \mathbb{R}^M$ is the weight vector. PA first initializes the weight to zero vector ($\mathbf{w}_1 = 0$) and, after observing the t-th example, the new weight $\mathbf{w}_{t+1}$ is obtained by minimizing the objective function $Q(\mathbf{w})$ defined as[1]

$$Q(\mathbf{w}) = \frac{1}{2}||\mathbf{w} - \mathbf{w}_t||^2 + C \cdot H\left(\mathbf{w}; (\mathbf{x}_t, y_t)\right) \quad (1)$$

[1]Here we study the PA-I version of PA.

where $H\left(\mathbf{w}; (\mathbf{x}_t, y_t)\right) = \max(0, 1 - y_t\mathbf{w}^T\mathbf{x}_t)$ is the *hinge loss* and C is the user-specified non-negative parameter. The intuitive goal of PA is to minimally change the existing weight $\mathbf{w}_t$ while making as accurate as possible prediction on the t-th example. Parameter C serves to balance these two competing objectives. Minimizing $Q(\mathbf{w})$ can be redefined as the following constrained optimization problem,

$$\begin{aligned} &\arg\min_{\mathbf{w}} \tfrac{1}{2}||\mathbf{w} - \mathbf{w}_t||^2 + C \cdot \xi \\ &\text{s.t. } 1 - y_t\mathbf{w}^T\mathbf{x}_t \leq \xi, \xi \geq 0, \end{aligned} \quad (2)$$

where the hinge loss is replaced with slack variable ξ and two inequality constraints.

The solution of (2) can be expressed in the closed-form as

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t\mathbf{x}_t, \;\; \alpha_t = y_t \cdot \min\left\{C, \frac{H\left(\mathbf{w}; (\mathbf{x}_t, y_t)\right)}{||\mathbf{x}_t||^2}\right\}. \quad (3)$$

It can be seen that $\mathbf{w}_{t+1}$ is obtained by adding the weighted new example only if the hinge loss of the current predictor is positive. Following the terminology of SVM, the examples with nonzero weights (i.e. $\alpha \neq 0$) are called the support vectors (SVs).

When using $\Phi(\mathbf{x})$ instead of $\mathbf{x}$ in the prediction model, where Φ denotes a mapping from the original input space $\mathbb{R}^M$ to the feature space $\mathbb{F}$, the PA predictor becomes able to solve nonlinear problems. Instead of directly computing and saving $\Phi(\mathbf{x})$, which might be infinitely dimensional, after introducing the Mercer kernel function k such that $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^T\Phi(\mathbf{x}')$, the model weight at the moment of receiving the t-th example, denoted as $\mathbf{w}_t$, can be represented by the linear combination of SVs,

$$\mathbf{w}_t = \sum_{i \in I_t} \alpha_i \Phi(\mathbf{x}_i), \quad (4)$$

where $I_t = \{i | \forall i < t, \alpha_i \neq 0\}$ is the set of indices of SVs. In this case, the resulting predictor, denoted as $f_t(\mathbf{x})$ can be represented as

$$f_t(\mathbf{x}) = \mathbf{w}_t^T\Phi(\mathbf{x}) = \sum_{i \in I_t} \alpha_t k(\mathbf{x}_t, \mathbf{x}). \quad (5)$$

While kernelization gives ability to solve nonlinear classification problems, it also increases the computational burden, both in time and space. In order to use the prediction model (4), all SVs and their weights should be stored in memory. The unbounded growth in the number of support vectors that is inevitable on noisy data implies unlimited memory budget and runtime. To solve this issue, we modify the PA algorithm such that the number of support vectors is strictly bounded by a predefined budget.

3 BUDGETED PA (BPA)

Let us assume the current weight $\mathbf{w}_t$ is composed of B support vectors, where B is the predefined budget. If the hinge loss on the newly received example $(\mathbf{x}_t, y_t)$ is positive, the budgeted PA (BPA) is attempting to achieve accurate prediction on the new example and produce new weight $\mathbf{w}_{t+1}$ that is 1) similar to $\mathbf{w}_t$, 2) has a small loss on t-th example and 3) spanned by only B of the available $B+1$ (because of the addition of the new example) SVs. This approach implies that one of the $B+1$ SVs can be removed. There are two decisions that BPA has to make: 1) which SV to remove and 2) what values should it assign to the remaining B SVs. We formalize the BPA problem and derive its solution in the rest of this section. In the following, we will replace input vectors $\mathbf{x}$ with feature vectors $\Phi(\mathbf{x})$ and will assume that $\mathbf{w}_t$ is defined as in (4).

The objective function of BPA is identical to that of PA (1). By assuming that SV $\mathbf{x}_r$, $r \in I_t \cup \{t\}$, has to be removed, BPA adds the following constraint to (2),

$$\mathbf{w} = \mathbf{w}_t - \alpha_r \Phi(\mathbf{x}_r) + \textstyle\sum_{i \in S} \beta_i \Phi(\mathbf{x}_i), \qquad (6)$$

where $S \subseteq I_t \cup \{t\} - \{r\}$ is a subset of the remaining SVs, after removal of SV $\mathbf{x}_r$. The specific choice of S is important from the BPA runtime perspective and will be discussed in detail in Section 3.1. For now, let us assume it has been specified by a user. The additional constraint enforces budget maintenance. It specifies that the new weight $\mathbf{w}$ updates the old weight $\mathbf{w}_t$ (first term in (6)) by deleting contribution of SV $\mathbf{x}_r$ (second term in (6)) and adding a new vector that is a linear combination of SVs in set S (third term in (6)), where $\beta_i, i \in S$, are the optimization parameters.

Let us denote $\mathbf{w}_{t+1}^r$ the solution of (2)+(6) when the SV $\mathbf{x}_r$ is removed. The BPA update is completed by finding $\mathbf{w}_{t+1}^r$ for each r and selecting the one that results in the minimal value of the objective function,

$$r^* = \underset{r \in I_t \cup \{t\}}{\arg\min}\, Q(\mathbf{w}_{t+1}^r). \qquad (7)$$

The optimal new weight is obtained as $\mathbf{w}_{t+1} = \mathbf{w}_{t+1}^{r^*}$ In the following proposition we give the closed-form solution of the problem (2)+(6), assuming r is known.

Proposition 1. *The solution of (2)+(6) is*

$$\mathbf{w}_{t+1}^r = \mathbf{w}_t - \alpha_r \Phi(\mathbf{x}_r) + \textstyle\sum_{i \in S} \beta_i \Phi(\mathbf{x}_i), \qquad (8)$$

where

$$\beta = \alpha_r \mathbf{K}^{-1}\mathbf{k}_r + \tau y_t \mathbf{K}^{-1}\mathbf{k}_t,$$
$$\tau = \min\left(C, \max(0, \frac{1 - y_t\left(f_t(\mathbf{x}_t) - \alpha_r k_{tr} + \alpha_r (\mathbf{K}^{-1}\mathbf{k}_r)^T \mathbf{k}_t\right)}{(\mathbf{K}^{-1}\mathbf{k}_t)^T \mathbf{k}_t})\right), \qquad (9)$$

$k_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, $\mathbf{K} = [k_{ij}], i, j \in S$ is the kernel matrix of SVs in S, $\mathbf{k}_i = [k_{ij}]^T, j \in S$ and $\beta = [\beta_i]^T, i \in S$.

Proof. We introduce the Lagrangian of the optimization problem and derive the solution that satisfies the KKT conditions. After replacing $\mathbf{w}$ with the right hand side of (6), Lagaragian of (2)+(6) can be expressed as

$$\begin{aligned} &L(\beta, \xi, \tau_1, \tau_2) \\ &= \tfrac{1}{2}||\sum_{i \in S} \beta_i \Phi(\mathbf{x}_i) - \alpha_r \Phi(\mathbf{x}_r)||^2 + C\xi + \tau_2(-\xi) \\ &+\tau_1(1 - y_t(\mathbf{w}_t - \alpha_r \Phi(\mathbf{x}_r) + \sum_{i \in S} \beta_i \Phi(\mathbf{x}_i))^T \Phi(\mathbf{x}_t) - \xi) \\ &= \tfrac{1}{2}\beta^T \mathbf{K}\beta - \alpha_r \beta^T \mathbf{k}_r + \tfrac{1}{2}\alpha_r^2 k_{rr} + C\xi \\ &+\tau_1\left(1 - y_t\left(f_t(\mathbf{x}_t) - \alpha_r k_{rt} + \beta^T \mathbf{k}_t\right) - \xi\right) - \tau_2 \xi, \end{aligned} \qquad (10)$$

where $\tau_1, \tau_2 \geq 0$ are the Lagrange multipliers. The optimal solution (β^*, ξ^*) should satisfy the following KKT conditions:

$$\begin{aligned} &\nabla_{\beta^*} L = 0, \frac{\partial L}{\partial \xi^*} = 0 \\ &\frac{\partial L}{\partial \tau_i} = 0,\ \tau_i \geq 0 \text{ if } g_i({\beta_t}^*, \xi^*) = 0,\ i = 1, 2 \\ &\frac{\partial L}{\partial \tau_i} = 0,\ \tau_i = 0 \text{ if } g_i({\beta_t}^*, \xi^*) < 0,\ i = 1, 2 \end{aligned} \qquad (11)$$

where $g_1(\beta^*, \xi^*) = 1 - y_t(f_t(\mathbf{x}_t) - \alpha_r k_{rt} + (\beta^*)^T \mathbf{k}_t) - \xi^*$ corresponds to the first inequality constraint and $g_2(\beta^*, \xi^*) = -\xi^*$ to the second inequality constraint of (2). After combining each solution with one of the four different cases in (11) and replacing τ_1 by τ we get the stated closed-form solution. □

It is worth to note that when $H(\mathbf{w}; (\Phi(\mathbf{x}_t), y_t)) = 0$, $\mathbf{w}_{t+1} = \mathbf{w}_t$ results in the minimal value 0 of the objective function $Q(\mathbf{w})$, which is equivalent to simply ignoring the t-th example. The runtime of (8) depends on the size of the kernel matrix $\mathbf{K}$ of S. In the following we present three BPA algorithms that have different computational cost. The proposed algorithms share a unified algorithmic structure which is summarized in Algorithm 1.

3.1 THREE VERSIONS

3.1.1 BPA-Simple (BPA-S)

In the simplest case, we assume that S only includes the new example t

$$S = \{t\}.$$

Therefore, only weight β_t of the new example is determined in (8), while the weights of all other SVs, except the removed one, remain the same. This idea of computing only β_t is consistent with the "quasi-additive"

Algorithm 1 Budgeted PA (BPA)

Input: Data stream D, kernel k, the regularization parameter C, budget B
Initialize: $\mathbf{w}_1 = 0$, $I_1 = \emptyset$
Output: $\mathbf{w}_{t+1}$
for $t = 1, 2, ...$ **do**
  observe $(\mathbf{x}_t, y_t)$;
  if $H(\mathbf{w}_t; (\Phi(\mathbf{x}_t), y_t)) = 0$ **then**
    $\mathbf{w}_{t+1} = \mathbf{w}_t$;
  else if $|I_t| < B$ **then**
    update $\mathbf{w}_{t+1}$ as in (3); //replace $\mathbf{x}$ by $\Phi(\mathbf{x})$
    $I_{t+1} = I_t \cup \{t\}$;
  else
    for $\forall r \in I_t \cup \{t\}$ **do**
      find $\mathbf{w}^r_{t+1}$ by (8);
    end for
    find r^* by (7);
    $\mathbf{w}_{t+1} = \mathbf{w}^{r*}_{t+1}$;
    $I_{t+1} = I_t \cup \{t\} - \{r^*\}$;
  end if
end for

updating style in the original memory-unbounded PA algorithm. Because in (8) $\mathbf{K}$ is replaced with k_{tt} and $\mathbf{k}_r$ with k_{rt} the optimal β_t is simplified to

$$\beta_t = \frac{\alpha_r k_{rt}}{k_{tt}} + \tau y_t, \text{where } \tau = \min\left(C, \frac{H(\mathbf{w}_t; (\mathbf{x}_t, y_t))}{k_{tt}}\right). \quad (12)$$

We name the resulting budgeted PA algorithm BPA-S. Comparing Eq. (12) with the original PA updating rule (3), it is interesting to see that β_t of BPA-S includes both the weighted coefficient of the removed example and the quantity calculated by the original PA update.

Cost The computation of each BPA-S update requires one-time computation of τ which costs $O(B)$. $Q(\mathbf{w}^r_{t+1})$ from (5) can be calculated in constant $O(1)$ time, so the evaluation of (7) has $O(B)$ cost. Therefore, the total cost of BPA-S update is $O(B)$. Since only B SVs and their α parameters are saved, the space complexity is also $O(B)$.

3.1.2 BPA-Projecting (BPA-P)

A benefit of the BPA-S updating strategy is on the computational side. On the other hand, weighting only the newest example is suboptimal. The best solution can be obtained if parameters β are calculated for all available SVs,

$$S = I_t + \{t\} - \{r\}.$$

We call the resulting budgeted PA algorithm BPA-P.

Cost BPA-P calculation (8) is dominated by the need to calculate $\mathbf{K}^{-1}$ for every r. Using the rank-one update (Cauwenberghs and Poggio, 2000), and observing that kernel matrices $\mathbf{K}$ for two different choices of r differ by only one row and column, $\mathbf{K}^{-1}$ for one r can be calculated using $\mathbf{K}^{-1}$ for another r in $O(B^2)$ time. Considering the need to apply (8) for every r, the total runtime of BPA-P for a single update is $O(B^3)$. The space scaling is $O(B^2)$.

3.1.3 BPA-Nearest-Neighbor (BPA-NN)

BPA-S gains in efficiency by calculating the coefficient of only the newest example, while BPA-P trades the efficiency with quality by updating the coefficients of all SVs, excluding the removed one. Here, we propose a middle-ground solution that approximates BPA-P and is nearly as fast as BPA-S. Taking a look at the calculation of β in (9) we can conclude that the r-th SV which is being removed is projected to the SVs indexed in S. In BPA-NN we define projection space as the union of the newest example and the n nearest neighbors[2] of the r-th SV. Since the nearest neighbors are most useful in projection, BPA-NN is a good approximation of the optimal BPA-P solution. In this paper we use only the nearest neighbor, $n = 1$,

$$S = \{t\} \cup NN(r)$$

where $NN(r)$ is the index of the nearest neighbor of $\mathbf{x}_r$. Accordingly, the model update is obtained by replacing $\mathbf{K}$ with

$$\begin{pmatrix} k_{NN(r),NN(r)} & k_{NN(r),t} \\ k_{NN(r),t} & k_{tt} \end{pmatrix}$$

and $\mathbf{k}_r$ with $[k_{r,NN(r)}\ k_{r,t}]^T$.

Cost At the first glance, finding the *nearest neighbor* for r-th SV would take $O(B)$ time. However, after using some bookkeeping by saving the indexes of nearest neighbors and the corresponding distances for each SV, the time complexity of finding k-NN could be reduced to $O(1)$. Thus, the computation of (8) takes $O(1)$ time. After deciding which example should be removed, the bookkeeping information is updated accordingly. Since, in average, each example is the nearest neighbor of one example, the removal of one SV costs the expected $O(B)$ time to update the bookkeeping information. Therefore, the whole updating procedure takes the expected $O(B)$ time and requires $O(B)$ space.

4 FROM HINGE TO RAMP LOSS

There are two practical issues with hinge loss that are relevant to the design of budgeted PA algorithms. The

[2]Calculated as Euclidean distance.

first is scalability. With hinge loss, all misclassified ($y_i f(\mathbf{x}_i) < 0$) and low-confidence correctly classified ($0 \le y_i f(\mathbf{x}_i) < 1$) examples become part of the model. As a consequence, the number of SVs increases with the training size in the PA algorithm. This scalability problem is particularly pronounced when there is a strong overlap between classes in the feature space. The second problem is that, with hinge loss, noisy and outlying examples have large impact on the cost function (1). This can negatively impact accuracy of the resulting PA predictor. To address these issues, in this section we show how the *ramp loss* (Collobert, et al., 2006)

$$R(\mathbf{w};(\mathbf{x}_t,y_t)) = \begin{cases} 0, & y_i\mathbf{w}^T\mathbf{x}_i > 1 \\ 1 - y_i\mathbf{w}^T\mathbf{x}_i, & -1 \le y_i\mathbf{w}^T\mathbf{x}_i \le 1 \\ 2, & y_i\mathbf{w}^T\mathbf{x}_i < -1, \end{cases} \tag{13}$$

can be employed in the both standard and budgeted versions of PA algorithm.

4.1 RAMP LOSS PA (PA^R)

Replacing the hinge loss H with the ramp loss R in (1), the ramp loss PA (PA^R) optimization problem reads as

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w} - \mathbf{w}_t||^2 + \mathrm{C} \cdot R(\mathbf{w};(\mathbf{x}_t,y_t)) \tag{14}$$

Because of the ramp loss function, the above objective function becomes non-convex. The following proposition will give us the solution of (14).

Proposition 2. *The solution of the non-convex optimization problem (14) leads to the following update rule*

$$\mathbf{w} = \mathbf{w}_t + \alpha_t\mathbf{x}_t, \quad \text{where}$$
$$\alpha_t = \begin{cases} \min\left\{C, \frac{H(\mathbf{w}_t;(\mathbf{x}_t,y_t))}{||\mathbf{x}_t||^2}\right\} & \text{if } |f_t(\boldsymbol{x}_t)| \le 1 \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

Proof. To solve problem (14) we use Convex Procedure (CCCP). CCCP (Yuille and Rangarajan, 2002) is an iterative procedure that solves non-convex optimization problem by solving a series of decomposed convex approximations. In our case, (14) is decomposed into the convex part $J_{vex}(\mathbf{w})$ and the concave part $J_{cave}(\mathbf{w})$ as follows

$$\begin{aligned} &\min_{\mathbf{w}} \tfrac{1}{2}||\mathbf{w} - \mathbf{w}_t||^2 + CR(\mathbf{w};(\mathbf{x}_t,y_t)) \\ &= \underbrace{\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w} - \mathbf{w}_t||^2 + CH(\mathbf{w};(\mathbf{x}_t,y_t))}_{J_{vex}(\mathbf{w})} + \\ &\underbrace{C\left(R(\mathbf{w};(\mathbf{x}_t,y_t)) - H(\mathbf{w};(\mathbf{x}_t,y_t))\right)}_{J_{cave}(\mathbf{w})}. \end{aligned} \tag{16}$$

After initializing $\mathbf{w}_{tmp}$ to $\mathbf{w}_t$, CCCP iteratively updates $\mathbf{w}_{tmp}$ by solving the following convex approximation problem

$$\begin{aligned} \mathbf{w}_{tmp} &= \arg\min_{\mathbf{w}} \left(J_{vex}(\mathbf{w}) + (\nabla_{\mathbf{w}_{tmp}} J_{cave})^T\mathbf{w}\right) \\ &= \begin{cases} \arg\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w} - \mathbf{w}_t||^2 + CH(\mathbf{w};(\mathbf{x}_t,y_t)), \\ \qquad\qquad\qquad\qquad \text{if} y_t f_{tmp}(\mathbf{x}) \ge -1 \\ \arg\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w} - \mathbf{w}_t||^2 + C, \quad \text{otherwise.} \end{cases} \end{aligned} \tag{17}$$

The procedure stops if $\mathbf{w}_{tmp}$ does not change.

Next we show that this procedure converges after a single iteration. First, suppose $y_t f_{tmp}(\mathbf{x}) \ge -1$. Then (17) is identical to problem (2). Thus, $\mathbf{w}_{tmp}$ is updated as (3) and in the next iteration we get

$$\begin{aligned} y_t f_{tmp}(\mathbf{x}_t) &= y_t(\mathbf{w}_t + \alpha_t\mathbf{x}_t)^T\mathbf{x}_t \\ &= y_t f_t(\mathbf{x}_t) + y_t\alpha_t||\mathbf{x}_t||^2 \ge -1, \end{aligned}$$

which leads the procedure to arrive at the same $\mathbf{w}_{tmp}$ as in the previous iteration; thus the CCCP terminates. Next, let us assume $y_t f_{tmp}(\mathbf{x}) < -1$, in which it is easy to see that the optimal solution is $\mathbf{w}_{tmp} = \mathbf{w}_t$. Since $\mathbf{w}_{tmp}$ remains unchanged, the procedure converges. According to the above analysis we conclude that CCCP converges after the first iteration. Combining with the fact that hinge loss equals zero when $y_t f_t(\mathbf{x}_t) > 1$ we get the stated update rule. □

4.2 RAMP LOSS BUDGETED PA (BPA^R)

By replacing the hinge loss with the ramp loss for the budgeted algorithms in Section 3, we obtain the budgeted ramp loss PA (BPA^R) algorithms. The resulting optimization problem is similar to the hinge loss version (as in the budgeted hinge loss PA case, here we will replace $\mathbf{x}$ with $\Phi(\mathbf{x})$),

$$\begin{aligned} &\min_{\mathbf{w}} \tfrac{1}{2}||\mathbf{w} - \mathbf{w}_t||^2 + \mathrm{C}R(\mathbf{w};(\Phi(\mathbf{x}_t),y_t)) \\ &\text{s.t.} \quad \mathbf{w} = \mathbf{w}_t - \alpha_r\Phi(\mathbf{x}_r) + \textstyle\sum_{i\in S}\beta_i\Phi(\mathbf{x}_i), \end{aligned} \tag{18}$$

where r is assumed fixed, α_t is initialized to zero and

$$S = \begin{cases} \{t\} & \text{for BPA}^R\text{-S} \\ I_t + \{t\} - \{r\} & \text{for BPA}^R\text{-P} \\ \{t\} \cup NN(r) & \text{for BPA}^R\text{-NN.} \end{cases}$$

Proposition 3. *The solution of the non-convex optimization problem (18) leads to the following update rule*

$$\begin{aligned} &\mathbf{w} = \mathbf{w}_t - \alpha_r\Phi(\mathbf{x}_r) + \textstyle\sum_{i\in S}\beta_i\Phi(\mathbf{x}_i), \text{where} \\ &\beta = \begin{cases} \alpha_r\mathbf{K}^{-1}\mathbf{k}_r + \tau y_t\mathbf{K}^{-1}\mathbf{k}_t, & \text{if } |f_t(\boldsymbol{x}_t)| \le 1 \\ \mathbf{0}, & \text{otherwise,} \end{cases} \end{aligned} \tag{19}$$

and τ is defined as in (9).

Proof. The CCCP decomposition of the objective function of (18) is identical to (16) and, accordingly, the resulting procedure iteratively solves the approximation optimization problem (17) with one added constraint, namely,

$$\begin{aligned} &\mathbf{w}_{tmp} = \min_{\mathbf{w}} \left(J_{vex}(\mathbf{w}) + (\nabla_{\mathbf{w}_{tmp}} J_{cave})^T \mathbf{w} \right) \\ &\text{s.t. } \mathbf{w} = \mathbf{w}_t - \alpha_r \Phi(\mathbf{x}_r) + \textstyle\sum_{i \in S} \beta_i \Phi(\mathbf{x}_i). \end{aligned} \tag{20}$$

Next we discuss the convergence of CCCP. First, suppose $y_t f_{tmp}(\mathbf{x}_t) < -1$. Then the optimal solution is $\mathbf{w}_{tmp} = \mathbf{w}_t$. Since $\mathbf{w}_{tmp}$ is unchanged and the procedure converges. Next, suppose $y_t f_{tmp}(\mathbf{x}_t) \geq -1$. Then the procedure solves the first case its hinge loss counterpart problem (2)+(6) and $\mathbf{w}_{tmp}$ is updated as (8). Since the optimal solution decreases the objective function in (20) and because $H(\mathbf{w}_t; (\Phi(\mathbf{x}_t), y_t)) \leq 1$, we get

$$\begin{aligned} &\tfrac{1}{2}||\mathbf{w}_{tmp} - \mathbf{w}_t||^2 + \mathrm{C}H(\mathbf{w}_{tmp}; (\Phi(\mathbf{x}_t), y_t)) \\ &\leq \tfrac{1}{2}||\mathbf{w}_t - \mathbf{w}_t||^2 + \mathrm{C}H(\mathbf{w}_t; (\Phi(\mathbf{x}_t), y_t)) \leq C, \end{aligned}$$

which leads to $H(\mathbf{w}_{tmp}; (\Phi(\mathbf{x}_t), y_t)) \leq 1$ and thus $y_t f_{tmp}(\mathbf{x}_t) \geq -1$. Therefore, after finishing the first iteration the procedure converges and we arrive at the stated solution. □

4.3 RAMP LOSS BPA AS ACTIVE LEARNING ALGORITHMS

The only difference between hinge loss PA and ramp loss PA is that ramp loss updates the model only if the new example is within the margin (i.e.$|f_t(\mathbf{x}_t)| \leq 1$). This means that the decision to update the model can be given before observing label of the new example. Therefore, ramp loss PA can be interpreted as the active learning algorithm, similar to what has been already observed (Guillory et al., 2009; Wang and Vucetic, 2009).

5 EXPERIMENTS

Datasets. The statistics (size, dimensionality and percentage of majority class) of 7 data sets used in the experiments are summarized in the first row of Table 1. The multi-class data sets were converted to two class sets as in (Wang and Vucetic, 2010). For speech phoneme data set *Phoneme* the first 18 classes were separated from the remaining 23 classes. *NCheckerboard* is noisy version of *Checkerboard*, where class assignment was switched for 15% of the randomly selected examples. We used the noise-free *Checkerboard* as the test set. Attributes in all data sets were scaled to mean 0 and standard deviation 1.

Algorithms. On the non-budgeted algorithms side, we compared with *PA* (Crammer et al., 2006) and Kernel *Perceptron* (Rosenblatt, 1958). For the Budgeted algorithms, we compared with *Stoptron* (Orabona et al. 2008), the baseline budgeted kernel perceptron algorithm that stops updating the model when the budget is exceeded; *Random* budgeted kernel perceptron (Cesa-Bianchi and Gentile 2007) that randomly removes an SV to maintain the budget; *Forgetron* (Dekel et al. 2008), the budgeted kernel perceptron that removes the oldest SV to maintain the budget; *PA+Rand*, the baseline budgeted PA algorithm that removes an random SV when the budget is exceeded; and *Projectron++* (Orabona et al. 2008). For Projectron++, the number of SVs depends on a pre-defined model sparsity parameter. In our experiments, we set this parameter such that the number of SVs is equal to B at the end of training.

Hyperparameter tuning. RBF kernel, $k(\mathbf{x}, \mathbf{x}') = \exp(-||x-x'||2/2\sigma^2)$, was used in all experiments. The hyper-parameters C and σ^2 were selected using cross validation for all combinations of algorithm, data set and budget.

Results on Benchmark Datasets. Experimental results on 7 datasets with $B = 100, 200$ are shown in Table 1. Algorithms are grouped with respect to the update runtime. Accuracy on the separate test set is reported. Each result is based on 10 repetitions on randomly shuffled training data streams. The last column is the averaged accuracy of each algorithm on 7 data sets. For each combination of data set and budget, two budgeted algorithms with the best accuracy are in bold. The best accuracy of the non-budgeted algorithms is also in bold.

Comparison of non-budgeted kernel perceptron and PA from Table 1 shows that PA has significantly larger accuracy than perceptron on all data sets. Comparing the hinge loss and ramp loss non-budgeted PA, we can see the average accuracy of PA^R is slightly better than PA. In three of the noisiest datasets *Adult*, *Banana* and *NCheckerboard*, PA^R is more robust and produces sparser predictors than PA. Moreover, it is worth reminding that, unlike PA, PA^R does not require labels of examples with confident predictions (i.e. $|f(\mathbf{x})| > 1$), which might be an important advantage in the active learning scenarios.

On the budgeted algorithm side, BPA^R-P achieves the best average accuracy for both B values. It is closely followed by its hinge loss counterpart BPA-P. However, it should be emphasized that their update cost and memory requirements can become unacceptable when large budgets are used. In the category of the fast $O(B)$ algorithms, BPA^R-NN and BPA-NN are the most successful and are followed by BPA-S and BPA^R-S. The accuracy of BPA-NN is consistently higher than BPA-S on all data sets and is most often quite com-

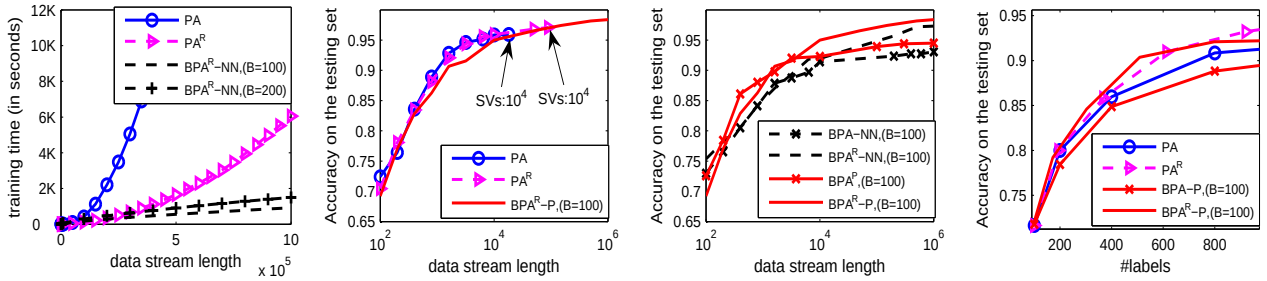
Table 1: Results on 7 benchmark datasets

Time	Algs	Adult 21K×123 75%	Banana 4.3K×2 55%	Checkerb 10K×2 50%	NCheckerb 10K×2 50%	Cover 10K×54 51%	Phoneme 10K×41 50%	USPS 7.3K×256 52%	Avg
Memory-unbounded online algorithms									
$O(N)$	Pcptrn	80.2±0.2	87.4±1.5	96.3±0.6	83.4±0.7	76.0±0.4	78.9±0.6	94.6±0.1	85.3
	(#SV)	(4.5K)	(0.6K)	(0.5K)	(2.8K)	(2.8K)	(2.4K)	(0.4K)	
	PA	83.6±0.2	89.1±0.7	97.2±0.1	95.8±1.0	81.6±0.2	82.6±0.9	96.7±0.1	89.5
	(#SV)	(15K)	(2K)	(2.6K)	(5.9K)	(9.9K)	(7.2K)	(4.5K)	
	PA^R	**84.1**±0.1	**89.3**±0.7	**97.5**±0.1	**96.2**±0.8	**82.7**±0.3	**83.7**±0.7	**96.7**±0.1	90.0
	(#SV)	(4.4K)	(1.5K)	(2.6K)	(3.3K)	(9.8K)	(6.5K)	(4.5K)	
Budgeted online algorithms (B=100)									
$O(B)$	Stptrn	76.5±2.0	86.7±2.1	87.3±0.9	75.4±4.3	64.2±1.7	67.6±2.7	89.1±1.2	78.1
	Rand	76.2±3.6	84.1±2.6	85.6±1.2	69.4±2.9	61.3±3.2	65.0±4.4	87.1±0.9	75.5
	Fogtrn	72.8±6.1	82.8±2.4	86.1±1.0	68.2±3.5	60.8±2.7	65.6±1.2	86.2±2.1	74.6
	PA+Rnd	78.4±1.9	84.9±2.1	83.3±1.4	75.1±3.6	63.1±1.5	64.0±3.9	86.2±1.1	76.4
	BPA-S	82.4±0.1	89.4±1.3	90.0±0.8	87.4±0.7	68.6±1.9	67.4±3.0	89.6±1.3	82.1
	BPA^R-S	82.4±0.1	89.5±1.7	90.0±1.0	88.2±1.2	69.3±1.8	67.0±3.2	89.3±1.2	82.2
	BPA-NN	82.8±0.4	89.6±1.4	94.0±1.2	90.2±1.3	69.1±1.8	74.3±0.7	90.8±0.9	84.4
	BPA^R-NN	**83.1**±0.0	89.8±1.1	94.2±0.9	**92.3**±0.5	70.3±0.8	74.6±0.8	90.8±0.6	85.0
$O(B^2)$	Pjtrn++	80.1±0.1	89.5±1.1	**95.4**±0.7	88.1±0.7	68.7±1.0	74.6±0.7	89.2±0.7	83.7
$O(B^3)$	BPA-P	83.0±0.2	**89.6**±1.1	**95.4**±0.7	91.7±0.8	**74.3**±1.4	**75.2**±1.0	**92.8**±0.7	86.0
	BPA-P^R	**84.0**±0.0	**89.6**±0.8	95.2±0.8	**94.1**±0.9	**75.0**±1.0	**74.9**±0.6	**92.6**±0.7	86.5
Budgeted online algorithms (B=200)									
$O(B)$	Stptrn	78.7±1.8	85.6±1.5	92.8±1.1	76.0±3.1	65.5±2.3	70.5±2.6	92.3±0.7	80.2
	Rand	76.4±2.8	83.6±2.0	90.3±1.3	74.5±2.1	62.4±2.4	67.3±2.5	89.8 ±1.1	77.8
	Fogtrn	72.9±6.8	85.0±1.3	90.9±1.7	72.2±4.4	62.1±2.8	68.0±2.3	90.3±0.9	77.3
	PA+Rnd	80.1±2.4	86.7±1.9	87.0±1.3	78.3±1.8	64.2±2.7	68.7±4.3	88.8±0.8	79.1
	BPA-S	82.7±0.2	89.5±0.7	93.4±0.5	89.7±0.9	71.7±1.7	71.3±2.3	92.6±0.9	84.4
	BPA^R-S	83.1±0.1	89.5±0.9	93.9±0.6	90.8±0.8	71.7±1.2	71.6±2.2	92.1±0.6	84.7
	BPA-NN	83.1±0.4	89.6±1.1	95.5±0.4	91.7±1.3	72.7±1.0	75.8±1.0	92.8±0.6	85.9
	BPA^R-NN	83.3±0.4	89.5±1.4	95.2±0.5	**93.3**±0.6	72.7±1.4	77.2±1.7	94.0±0.4	86.5
$O(B^2)$	Pjtrn++	82.9±0.1	89.5±1.2	**95.8**±0.5	92.5±1.0	75.1±2.0	75.2±0.6	93.2±0.6	86.3
$O(B^3)$	BPA-P	**83.8**±0.0	**89.7**±0.7	**95.9**±0.6	92.8±0.7	**76.0**±1.3	**78.0**±0.3	**94.8**±0.3	87.3
	BPA^R-P	**84.6**±0.0	**90.3**±1.5	95.6±1.2	**94.5**±1.1	**76.3**±1.0	**77.6**±0.6	**94.8**±0.3	87.7

petitive to BPA-P. This clearly illustrates the success of the nearest neighbor strategy. Interestingly, BPA^R-NN is more accurate than the recently proposed Projectron++, despite its significantly lower cost. The baseline budgeted PA algorithm PA+Rand is less successful than any of the proposed BPA algorithms. The performance of the perceptron-based budgeted algorithms Stoptron, Random and Forgetron is not impressive.

As the budget increases from 100 to 200, the accuracy of all budgeted algorithms is improved towards the accuracy of non-budgeted algorithms. *Covertype* and *Phoneme* are the most challenging data sets for budgeted algorithms, because they represent highly complex concepts and have a low level of noise. Considering that more than 98% and 72% of examples in these two data sets become SVs in PA, the accuracy of online budgeted algorithms with small budgets of $B = 100$ or 200 is impressive. It is clear that higher budget would result in further improvements in accuracy of the budgeted algorithms.

Detailed Results on 1 Million Examples. In this experiment, we illustrate the superiority of the budgeted online algorithms over the non-budget ones on a large-scale learning problem. A data set of 1 million *NCheckerboard* examples was used. The non-budgeted algorithms PA, PA^R and four most successful budgeted algorithms from Table 1, BPA-NN, BPA^R-NN, BPA-P and BPA^R-P were used in this experiment. The hyper-parameter tuning was performed on a subset of 10,000 examples. Considering the long runtime of PA and PA^R, they were early stopped when the number of SVs exceeded 10,000. From Figure 2(a) we can observe that PA has the fastest increase in training time. The training time of both non-budgeted algorithms grows quadratically with the training size, as expected. Unlike them, runtime of BPA^R-NN indeed scales linearly. BPA^R-NN with $B = 200$ is about two times slower than with $B = 100$, which confirms that its runtime scales only linearly with B. Figures 2(b) and 2(c) show the accuracy evolution curves. From Figure 2 (b) we can see the accuracy curve of BPA^R-P steadily increased during the online process and even-



(a) (b) (c) (d)

Figure 1: Trained on 1 million examples

tually outperformed PA and PA^R. In Figure 2(c), BPA^R-NN is quite competitive to BPA^R-P and all algorithms have the consistently growing accuracy. In Figure 2 (d), the number of labels queried versus the accuracy is plotted for four algorithms. It can be seen that for the fixed number of queried labels the ramp loss PA algorithms PA^R and BPA^R-P achieved better accuracy than the hinge-loss algorithms that require labels for each example. Interestingly, BPA^R-P was slightly more successful than PA^R at initial stage.

6 CONCLUSION

In this paper, we proposed a family of budgeted PA algorithms. To maintain the budget, a joint optimization problem is solved with respect to different updating strategies and loss functions. The resulting algorithms share a unified update rule but have different time and space costs. Experimental results show that 1) the proposed budget algorithms significantly improve accuracy over the previously proposed budgeted algorithms; 2) ramp loss based algorithms are more robust to noisy data and can produce sparser models; 3) ramp loss PA can be directly interpreted as active learning algorithm; and 4) the proposed budgeted algorithms are applicable for large-scale learning. Finally, the idea of budgeted PA can be extended beyond binary classification, but this is left for the future work.

Acknowledgements

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

References

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

N. Cesa-Bianchi and C. Gentile (2007). Tracking the best hyperplane with a simple budget perceptron. *MLJ*.

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 and V. Vapnik (1995). Support-vector networks. *MLJ*.

K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz and Y. Singer (2006). Online passive-aggressive algorithms. *JMLR*.

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

O. Dekel and S. S. Shwartz and Y. Singer (2008). The Forgetron: a kernel-based Perceptron on a budget. *SIAM Journal on Computing*.

A. Guillory, E. Chastain and J. Bilmes (2009). Active learning as nonconvex optimization. *AISTATS*.

F. Orabona, J. Keshet and B. Caputo (2008). The Projectron: a bounded kernel-based perceptron. *ICML*.

F. Rosenblatt (1958). The perceptron: a probabilistic the brain. *Psychological Review*.

S. Vucetic, V. Coric and Z. Wang (2009). Compressed kernel perceptrons. *DCC*.

Z. Wang and S. Vucetic (2009). Fast online training of ramp loss support vector machines. *ICDM*.

Z. Wang and S. Vucetic (2010). Twin vector machines for online learning on a budget. To appear in *Statistical Analysis and Data Mining Journal*.

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

A. L. Yuille and A. Rangarajan (2002). The concave convex procedure (CCCP). *NIPS*.