BDUOL: Double Updating Online Learning on a Fixed Budget

Peilin Zhao and Steven C.H. Hoi

School of Computer Engineering,
Nanyang Technological University, Singapore
E-mail: {zhao0106,chhoi}@ntu.edu.sg

Abstract. Kernel-based online learning often exhibits promising empirical performance for various applications according to previous studies. However, it often suffers a main shortcoming, that is, the unbounded number of support vectors, making it unsuitable for handling large-scale datasets. In this paper, we investigate the problem of budget kernel-based online learning that aims to constrain the number of support vectors by a predefined budget when learning the kernel-based prediction function in the online learning process. Unlike the existing studies, we present a new framework of budget kernel-based online learning based on a recently proposed online learning method called "Double Updating Online Learning" (DUOL), which has shown state-of-the-art performance as compared with the other traditional kernel-based online learning algorithms. We analyze the theoretical underpinning of the proposed Budget Double Updating Online Learning (BDUOL) framework, and then propose several BDUOL algorithms by designing different budget maintenance strategies. We evaluate the empirical performance of the proposed BDUOL algorithms by comparing them with several well-known budget kernel-based online learning algorithms, in which encouraging results validate the efficacy of the proposed technique.

1 Introduction

The goal of kernel-based online learning is to incrementally learn a nonlinear kernel-based prediction function from a sequence of training instances [1–4]. Although it often yields significantly better performance than linear online learning, the main shortcoming of kernel-based online learning is its potentially unbounded number of support vectors with the kernel-based prediction function, which thus requires a large amount of memory for storing support vectors and a high computational cost of making predictions at each iteration, making it unsuitable for large-scale applications. In this paper, we aim to tackle this challenge by studying a framework for kernel-based online learning on a fixed budget or known as "budget online learning" for short, in which the number of support vectors for the prediction function is bounded by some predefined budget size.

In literature, several algorithms have been proposed for budget online learning. Crammer et al. [5] proposed a heuristic approach for budget online learning

by extending the classical kernel-based perceptron method [6], which was further improved in [7]. The basic idea of these two algorithms is to remove the support vector that has the least impact on the classification performance whenever the budget, i.e., the maximal number of support vectors, is reached. The main shortcoming of these two algorithms is that they are heuristic without solid theoretic supports (e.g., no any mistake/regret bound was given).

Forgetron [8] is perhaps the first approach for budget online learning that offers a theoretical bound of the total number of mistakes. In particular, at each iteration, if the online classifier makes a mistake, it conducts a three-step updating: (i) it first runs the standard Perceptron [6] for updating the prediction function; (ii) it then shrinks the weights of support vectors by a carefully chosen scaling factor; and (iii) it finally removes the support vector with the least weight. Another similar approach is the Randomized Budget Perceptron (RBP) [9], which randomly removes one of existing support vectors when the number of support vectors exceeds the predefined budget. In general, RBP achieves similar mistake bound and empirical performance as Forgetron.

Unlike the above strategy that discards one support vector to maintain the budget, Projectron [10] adopts a projection strategy to bound the number of support vectors. Specifically, at each iteration where a training example is misclassified, it first updates the kernel classifier by applying a standard Perceptron; it then projects the new classifier into the space spanned by all the support vectors except the new example received at the current iteration, if the difference between the new classifier and its projection is less than a given threshold, otherwise it will remain unchanged. Empirical studies show that Projectron usually outperforms Forgetron in classification but with significantly longer running time. In addition to its high computational cost, another shortcoming of Projectron is that although the number of support vectors is bounded, it is unclear the exact number of support vectors achieved by Projectron in theory.

The above budget online learning approaches were designed based on the Perceptron learning framework [6]. In this paper, we propose a new framework of Budget Double Updating Online Learning (BDUOL) based on a recently proposed Double Updating Online Learning (DUOL) technique [4], which has shown state-of-the-art performance for online learning. The key challenge is to develop an appropriate strategy for maintaining the budget whenever the size of support vectors overflows. In this paper, following the theory of double updating online learning, we analyze the theoretical underpinning of the BDUOL framework, and propose a principled approach to developing three different budget maintenance strategies. We also analyze the mistake bounds of the proposed BDUOL algorithms and evaluate their empirical performance extensively.

The rest of the paper is organized as follows. Section 2 first introduces the problem setting and then presents both theoretical and algorithmic framework of the proposed budget online learning technique. Section 3 presents several different budget maintenance strategies for BDUOL. Section 4 discusses our empirical studies. Section 5 concludes this work.

2 Double Updating Online Learning on A Fixed Budget

In this section, we first introduce the problem setting for online learning and Double Updating Online Learning (DUOL), and then present the details of the proposed Budget Double Updating Online Learning framework.

2.1 Problem Setting

We consider the problem of online classification on a fixed budget. Our goal is to learn a prediction function $f : \mathbb{R}^d \to \mathbb{R}$ from a sequence of training examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$, where $\mathbf{x}_t \in \mathbb{R}^d$ is a d-dimensional instance and $y_t \in \mathcal{Y} = \{-1, +1\}$ is the class label assigned to $\mathbf{x}_t$. We use $sign(f(\mathbf{x}))$ to predict the class assignment for any $\mathbf{x}$, and $|f(\mathbf{x})|$ to measure the classification confidence. Let $\ell(f(\mathbf{x}), y) : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$ be the loss function that penalizes the deviation of estimates $f(\mathbf{x})$ from observed labels y. We refer to the output f of the learning algorithm as a *hypothesis* and denote the set of all possible hypotheses by $\mathcal{H} = \{f | f : \mathbb{R}^d \to \mathbb{R}\}$.

In this paper, we consider $\mathcal{H}$ a Reproducing Kernel Hilbert Space (**RKHS**) endowed with a kernel function $\kappa(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ [11] implementing the inner product$\langle \cdot, \cdot \rangle$ such that: 1) reproducing property $\langle f, \kappa(\mathbf{x}, \cdot) \rangle = f(\mathbf{x})$ for $\mathbf{x} \in \mathbb{R}^d$; 2) $\mathcal{H}$ is the closure of the span of all $\kappa(\mathbf{x}, \cdot)$ with $\mathbf{x} \in \mathbb{R}^d$, that is, $\kappa(\mathbf{x}, \cdot) \in \mathcal{H}$ for every $\mathbf{x} \in \mathcal{X}$. The inner product $\langle \cdot, \cdot \rangle$ induces a norm on $f \in H$ in the usual way: $\|f\|_{\mathcal{H}} := \langle f, f \rangle^{\frac{1}{2}}$. To make it clear, we use $\mathcal{H}_\kappa$ to denote an RKHS with explicit dependence on kernel function κ. Throughout the analysis, we assume $\kappa(\mathbf{x}, \mathbf{x}) \leq 1$ for any $\mathbf{x} \in \mathbb{R}^d$.

2.2 Double Updating Online Learning: A Review

Our BDUOL algorithm is designed based on the state-of-the-art Double Updating Online Learning (DUOL) method [4]. Unlike traditional online learning algorithms that usually perform a single update for each misclassified example, DUOL not only updates the weight for the newly added Support Vector (SV), but also updates that of another existing SV, which conflicts most with the new SV. Furthermore, both theoretical and empirical analysis have demonstrated the effectiveness of this algorithm. Below we briefly review the basics of double updating online learning.

Consider an incoming instance $\mathbf{x}_t$ received at the t-th step of online learning. The algorithm predicts the class label $\widehat{y}_t = \text{sgn}(f_{t-1}(\mathbf{x}_t))$ using the following kernel-based classifier:

$$f_{t-1}(\cdot) = \sum_{i \in S_{t-1}} \widehat{\gamma}_i y_i \kappa(\mathbf{x}_i, \cdot),$$

where S_{t-1} is the index set of the SVs for the $(t-1)$-th step, and $\widehat{\gamma}_i$ is the weight of the i-th existing support vector. After making the prediction, the algorithm will suffer a loss, defined by a hinge loss as $\ell(f_{t-1}(\mathbf{x}_t), y_t) = \max(0, 1 - y_t f_{t-1}(\mathbf{x}_t))$. If $\ell(f_{t-1}(\mathbf{x}_t), y_t) > 0$, the DUOL algorithm will update the prediction function f_{t-1} to f_t by adding the training example $(\mathbf{x}_t, y_t)$ as a new support vector.

Specifically, when the new added example $(\mathbf{x}_t, y_t)$ conflicts with $(\mathbf{x}_b, y_b)$, $b \in S_{t-1}$, by satisfying conditions: 1)$\ell_t = 1 - y_t f_{t-1}(\mathbf{x}_t) > 0$; 2) $\ell_b = 1 - y_b f_{t-1}(\mathbf{x}_b) > 0$; 3) $y_t y_b \kappa(\mathbf{x}_t, \mathbf{x}_b) \leq \min(-\rho, y_t y_a \kappa(\mathbf{x}_t, \mathbf{x}_a))$, $a \in S_{t-1}$, and $a \neq b$, where $\rho \in [0, 1)$ is a threshold, then the updating strategy referred as double updating will be adopted as follows:

$$f_t(\cdot) = f_{t-1}(\cdot) + \gamma_t y_t \kappa(\mathbf{x}_t, \cdot) + d_{\gamma_b} y_b \kappa(\mathbf{x}_b, \cdot),$$

where γ_t and d_{γ_b} are computed in the following equations:

$$(\gamma_t, d_{\gamma_b}) = \begin{cases} (C, C - \hat{\gamma}_b) & \text{if } (k_t C + w_{ab}(C - \hat{\gamma}_b) - \ell_t) < 0 \text{ and} \\ & \quad (k_b(C - \hat{\gamma}_b) + w_{ab} C - \ell_b) < 0 \\ (C, \frac{\ell_b - w_{ab} C}{k_b}) & \text{if } \frac{w_{ab}^2 C - w_{ab}\ell_b - k_t k_b C + k_b \ell_t}{k_b} > 0 \text{ and} \\ & \quad \frac{\ell_b - w_{ab} C}{k_b} \in [-\hat{\gamma}_b, C - \hat{\gamma}_b] \\ (\frac{\ell_t - w_{ab}(C - \hat{\gamma}_b)}{k_t}, C - \hat{\gamma}_b) & \text{if } \frac{\ell_t - w_{ab}(C - \hat{\gamma}_b)}{k_t} \in [0, C] \text{ and} \\ & \quad \ell_b - k_b(C - \hat{\gamma}_b) - w_{ab}\frac{\ell_t - w_{ab}(C - \hat{\gamma}_b)}{k_t} > 0 \\ (\frac{k_b \ell_t - w_{ab}\ell_b}{k_t k_b - w_{ab}^2}, \frac{k_t \ell_b - w_{ab}\ell_t}{k_t k_b - w_{ab}^2}) & \text{if } \frac{k_b \ell_t - w_{ab}\ell_b}{k_t k_b - w_{ab}^2} \in [0, C] \text{ and} \\ & \quad \frac{k_t \ell_b - w_{ab}\ell_t}{k_t k_b - w_{ab}^2} \in [-\hat{\gamma}_b, C - \hat{\gamma}_b] \end{cases}, \quad (1)$$

where $k_t = \kappa(\mathbf{x}_t, \mathbf{x}_t), k_b = \kappa(\mathbf{x}_b, \mathbf{x}_b)$, $w_{ab} = y_t y_b \kappa(\mathbf{x}_t, \mathbf{x}_b)$ and $C > 0$; or when no existing SV conflicts with $(\mathbf{x}_t, y_t)$, the single update strategy will be adopted as follows:

$$f_t(\cdot) = f_{t-1}(\cdot) + \gamma_t y_t \kappa(\mathbf{x}_t, \cdot), \quad (2)$$

where $\gamma_t = \min(C, \ell_t / k_t^2)$. It is not difficult to see that the single update strategy is reduced to the Passive-Aggressive updating strategy [3].

2.3 Framework of Budget Double Updating Online Learning

Although DUOL outperforms various traditional single updating algorithms, one major limitation is that it does not bound the number of support vectors, which could result in high computation and heavy memory cost when being applied to large-scale applications. In this paper, we aim to overcome this limitation by proposing a budget double updating online learning framework in which the number of support vectors is bounded by a predefined budget size.

Let us denote by B a predefined budget size for the maximal number of support vectors associated with the prediction function. The key difference of Budget DUOL over regular DUOL is to develop an appropriate budget maintenance step to ensure that the number of support vectors with the classifier f_t is less than the budget B at the beginning of each online updating step. In particular, let us denote by f_{t-1} the classifier produced by a regular DUOL at the $t-1$-th step, when the support vector size of f_{t-1} is equal to the B, BDUOL performs the classifier update towards budget maintenance: $f_{t-1} \leftarrow f_{t-1} - \Delta f_{t-1}$ such that the support vector size of the updated f_{t-1} is smaller than B. The details of the proposed Budget DUOL (BDUOL) algorithmic framework are summarized in Algorithm 1.

Algorithm 1 The Budget Double Updating Online Learning Algorithm (**BDUOL**)

PROCEDURE
Initialize $S_0 = \emptyset$, $f_0 = 0$;
for t=1,2,...,T **do**
  Receive a new instance $\mathbf{x}_t$;
  Predict $\hat{y}_t = \text{sign}(f_{t-1}(\mathbf{x}_t))$;
  Receive its label y_t;
  $\ell_t = max\{0, 1 - y_t f_{t-1}(\mathbf{x}_t)\}$;
  if $\ell_t > 0$ **then**
    if ($|S_t| == B$) **then**
      $f_{t-1} = f_{t-1} - \Delta f_{t-1}$; (Budget Maintenance)
    end if
    $\ell_t = max\{0, 1 - y_t f_{t-1}(\mathbf{x}_t)\}$;
    if $\ell_t > 0$ **then**
      $w_{min} = \infty$;
      for $\forall i \in S_{t-1}$ **do**
        if ($f^i_{t-1} \leq 1$) **then**
          if ($y_i y_t \kappa(\mathbf{x}_i, \mathbf{x}_t) \leq w_{min}$) **then**
            $w_{min} = y_i y_t \kappa(\mathbf{x}_i, \mathbf{x}_t)$;
            $(\mathbf{x}_b, y_b) = (\mathbf{x}_i, y_i)$;
          end if
        end if
      end for
      $f^t_{t-1} = y_t f_{t-1}(\mathbf{x}_t)$;
      $S_t = S_{t-1} \cup \{t\}$;
      if ($w_{min} \leq -\rho$) **then**
        Compute γ_t and d_{γ_b} using equation (1);
        for $\forall i \in S_t$ **do**
          $f^i_t \leftarrow f^i_{t-1} + y_i \gamma_t y_t \kappa(\mathbf{x}_i, \mathbf{x}_t) + y_i d_{\gamma_b} y_b \kappa(\mathbf{x}_i, \mathbf{x}_b)$;
        end for
        $f_t = f_{t-1} + \gamma_t y_t \kappa(\mathbf{x}_t, \cdot) + d_{\gamma_b} y_b \kappa(\mathbf{x}_b, \cdot)$;
      else /* no auxiliary example found */
        $\gamma_t = min(C, \ell_t / \kappa(\mathbf{x}_t, \mathbf{x}_t))$;
        for $\forall i \in S_t$ **do**
          $f^i_t \leftarrow f^i_{t-1} + y_i \gamma_t y_t \kappa(\mathbf{x}_i, \mathbf{x}_t)$;
        end for
        $f_t = f_{t-1} + \gamma_t y_t \kappa(\mathbf{x}_t, \cdot)$;
      end if
    else
      $f_t = f_{t-1}$; $S_t = S_{t-1}$;
      for $\forall i \in S_t$ **do**
        $f^i_t \leftarrow f^i_{t-1}$;
      end for
    end if
  end if
end for
return f_T, S_T
END

Fig. 1. The Algorithms of Budget Double Updating Online Learning (BDUOL).

The key challenge of BDUOL is to choose an appropriate reduction term Δf_{t-1} by a proper budget maintenance strategy, which can only meet the budget requirement but also minimize the impact of the reduction on the prediction performance. Unlike some existing heuristic budget maintenance approaches, in this paper, we propose a principled approach for developing several different budget maintenance strategies. Before presenting the detailed strategies, in the following, we analyze the theoretical underpinning of the proposed budget double updating online learning scheme, which is the theoretical foundation for the proposed budget maintenance strategies in Section 3.

2.4 Theoretical Analysis

In this section, we analyze the mistake bound of the proposed BDUOL algorithm. To simplify the analysis, the primal-dual framework is used to derive the mistake bound following the strategy of DUOL algorithm. Through this framework, we will show that the gap between the mistake bound of DUOL and BDUOL is bounded by the cumulative dual ascent induced by the function reduction, i.e., $\Delta f = \sum \Delta\gamma_i y_i \kappa(\mathbf{x}_i, \cdot)$. To facilitate the analysis, we firstly introduce the following lemma, which provides the dual objective function of the SVM.

Lemma 1. *The dual objective of* $\mathcal{P}_t(f) = \frac{1}{2}\|f\|_{\mathcal{H}_\kappa} + C\sum_{i=1}^{t} \ell(f(\mathbf{x}_i), y_i)$, $C > 0$ *is*

$$\mathcal{D}_t(\gamma_1, \ldots, \gamma_t) = \sum_{i=1}^{t} \gamma_i - \frac{1}{2}\|\sum_{i=1}^{t} \gamma_i y_i \kappa(\mathbf{x}_i, \cdot)\|^2_{\mathcal{H}_\kappa}, \quad \gamma_i \in [0, C], \tag{3}$$

where the relation between f *and* γ_i, $i = 1, \ldots, t$ *is* $f(\cdot) = \sum_{i=1}^{t} \gamma_i y_i \kappa(\mathbf{x}_i, \cdot)$.

According to the above lemma, after the t-th budget maintenance, the resultant dual ascent will be computed as follows:

Theorem 1 *The* ***D****ual* ***A****scent* $DA_t = \mathcal{D}_t(\gamma_1 - \Delta\gamma_1, \ldots, \gamma_t - \Delta\gamma_t) - \mathcal{D}_t(\gamma_1, \ldots, \gamma_t)$ *for the* t*-th budget maintenance, i.e.,* $f_t = f_t - \Delta f_t$*, is given as follows:*

$$DA_t = -\sum_{i=1}^{t} \Delta\gamma_i + \sum_{i=1}^{t} \Delta\gamma_i y_i f_t(\mathbf{x}_i) - \frac{1}{2}\|\Delta f_t\|^2_{\mathcal{H}_\kappa}. \tag{4}$$

Proof.

$$\begin{aligned}
&\mathcal{D}_t(\gamma_1 - \Delta\gamma_1, \ldots, \gamma_t - \Delta\gamma_t) - \mathcal{D}_t(\gamma_1, \ldots, \gamma_t) \\
&= \sum_{i=1}^{t}(\gamma_i - \Delta\gamma_i) - \frac{1}{2}\|\sum_{i=1}^{t}(\gamma_i - \Delta\gamma_i) y_i \kappa(\mathbf{x}_i, \cdot)\|^2_{\mathcal{H}_\kappa} - [\sum_{i=1}^{t} \gamma_i - \frac{1}{2}\|\sum_{i=1}^{t} \gamma_i y_i \kappa(\mathbf{x}_i, \cdot)\|^2_{\mathcal{H}_\kappa}] \\
&= -\sum_{i=1}^{t} \Delta\gamma_i + \frac{1}{2}[\|f_t\|^2_{\mathcal{H}_\kappa} - \|f_t - \Delta f_t\|^2_{\mathcal{H}_\kappa}] \\
&= -\sum_{i=1}^{t} \Delta\gamma_i + \frac{1}{2}[\|f_t\|^2_{\mathcal{H}_\kappa} - \|f_t\|^2_{\mathcal{H}_\kappa} + 2\langle f_t, \Delta f_t\rangle - \|\Delta f_t\|^2_{\mathcal{H}_\kappa}]
\end{aligned}$$

$$= -\sum_{i=1}^{t} \Delta\gamma_i + \langle f_t, \Delta f_t \rangle - \frac{1}{2}\|\Delta f_t\|^2_{\mathcal{H}_\kappa}$$
$$= -\sum_{i=1}^{t} \Delta\gamma_i + \langle f_t, \sum_{i=1}^{t} \Delta\gamma_i y_i \kappa(\mathbf{x}_i, \cdot)\rangle - \frac{1}{2}\|\Delta f_t\|^2_{\mathcal{H}_\kappa}$$
$$= -\sum_{i=1}^{t} \Delta\gamma_i + \sum_{i=1}^{t} \Delta\gamma_i y_i f_t(\mathbf{x}_i) - \frac{1}{2}\|\Delta f_t\|^2_{\mathcal{H}_\kappa}.$$

Based on the above dual ascent for budget maintenance, we can now analyze the mistake bound of the proposed BDUOL algorithm. To ease our discussion, we first introduce the following lemma [4] about the mistake bound of DUOL.

Lemma 2. *Let* $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)$ *be a sequence of examples, where* $\mathbf{x}_t \in \mathbb{R}^d$, $y_t \in \{-1, +1\}$ *and* $\kappa(\mathbf{x}_t, \mathbf{x}_t) \leq 1$ *for all* t*, and assume* $C \geq 1$*. Then for any function* f *in* $\mathcal{H}_\kappa$*, the number of prediction mistakes* M *made by DUOL on this sequence of examples is bounded by:*

$$2 \min_{f \in \mathcal{H}_\kappa} \left\{ \frac{1}{2}\|f\|^2_{\mathcal{H}_\kappa} + C \sum_{i=1}^{T} \ell(f(x_i), y_i) \right\} - \frac{\rho^2}{2} M_d^w(\rho) - \frac{1+\rho}{1-\rho} M_d^s(\rho),$$

where $\rho \in [0, 1)$*,* $M_d^w(\rho) > 0$ *and* $M_d^s(\rho) > 0$*.*

Combining the above lemma with Theorem 1, it is not difficult to derive the following mistake bound for the proposed BDUOL algorithm.

Theorem 2 *Let* $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)$ *be a sequence of examples, where* $\mathbf{x}_t \in \mathbb{R}^d$, $y_t \in \{-1, +1\}$ *and* $\kappa(\mathbf{x}_t, \mathbf{x}_t) \leq 1$ *for all* t*, and assume* $C \geq 1$*. Then for any function* f *in* $\mathcal{H}_\kappa$*, the number of prediction mistakes* M *made by BDUOL on this sequence of examples is bounded by:*

$$2 \min_{f \in \mathcal{H}_\kappa} \left\{ \frac{1}{2}\|f\|^2_{\mathcal{H}_\kappa} + C \sum_{i=1}^{T} \ell(f(x_i), y_i) \right\} - 2\sum_{i=1}^{T} DA_i - \frac{\rho^2}{2} M_d^w(\rho) - \frac{1+\rho}{1-\rho} M_d^s(\rho),$$

where $\rho \in [0, 1)$*.*

Proof. According to the proof of Lemma 2 [4], we have

$$\frac{1}{2}M_s + \frac{1+\rho^2}{2} M_d^w(\rho) + \frac{1}{1-\rho} M_d^s(\rho) \leq \min_{f \in \mathcal{H}_\kappa} \left\{ \frac{1}{2}\|f\|^2_{\mathcal{H}_\kappa} + C \sum_{i=1}^{T} \ell(f(x_i), y_i) \right\},$$

and $M = M_s + M_d^w(\rho) + M_d^s(\rho)$. Furthermore, by taking the dual ascents of budget maintenance into consideration, we have

$$\sum_{i=1}^{T} DA_i + \frac{M_s}{2} + \frac{1+\rho^2}{2} M_d^w(\rho) + \frac{M_d^s(\rho)}{1-\rho} \leq \min_{f \in \mathcal{H}_\kappa} \left\{ \frac{1}{2}\|f\|^2_{\mathcal{H}_\kappa} + C \sum_{i=1}^{T} \ell(f(x_i), y_i) \right\}.$$

Rearranging the above inequality will concludes the theorem.

According to the above theorem, we can see that, if there is no budget maintenance step, the mistake bound of BDUOL is reduced to the previous mistake bound for the regular DUOL algorithm. This theorem indicates that in order to minimize the mistake bound of BDUOL, one should try to maximize the cumulative dual ascent, i.e., $\sum_{i=1}^{T} DA_i$, when designing an appropriate budget maintenance strategy.

3 Budget Maintenance Strategies

In this section, we follow the above theoretical results to develop several different budget maintenance strategies in a principled approach. In particular, as revealed by Theorem 2, a key to improving the mistake bound of BDUOL is to maximize the cumulative dual ascent $\sum_{t=1}^{T} DA_t$ caused by the budget maintenance. To achieve this purpose, we propose to maximize the dual ascent caused by budget maintenance at each online learning step, i.e.,

$$\max_{\Delta\gamma_1,\ldots,\Delta\gamma_t} DA_t = -\sum_{i=1}^{t} \Delta\gamma_i + \sum_{i=1}^{t} \Delta\gamma_i y_i f_t(\mathbf{x}_i) - \frac{1}{2}\|\Delta f_t\|^2_{\mathcal{H}_\kappa}. \tag{5}$$

Below, we propose three different budget maintenance strategies, and analyze the principled approach of achieving the best dual ascent as well as the time complexity and memory cost for each strategy.

3.1 BDUOL Algorithm by Removal Strategy

The first strategy for budget maintenance is the removal strategy that discards one of existing support vectors, which is similar to the strategies used by Forgetron [8] and RBP [9]. Unlike the previous heuristic removal strategy, the key idea of our removal strategy is to discard the support vector which can maximize the dual ascent by following our previous analysis.

Specifically, let us assume the j-th SV is selected for removal. We then have the following function reduction term:

$$\Delta f_t = \gamma_j y_j \kappa(\mathbf{x}_j, \cdot). \tag{6}$$

As a result, the optimal removal solution is to discard the SV which can maximize the following dual ascent term:

$$DA_{t,j} = -\gamma_j(1 - y_j f_t(\mathbf{x}_j)) - \frac{1}{2}(\gamma_j)^2 \kappa(\mathbf{x}_j, \mathbf{x}_j). \tag{7}$$

We note that the above removal strategy is similar with the one in [5], when the Gaussian kernel is adopted where $\kappa(\mathbf{x}_j, \mathbf{x}_j) = 1$. However, our strategy is strongly theoretically motivated.

Complexity Analysis. Since the BDUOL algorithm will cache the value $y_j f_t(\mathbf{x}_j)$ for every SV, the computational complexity of $DA_{t,j}$ is $O(1)$ in practice. Thus, this BDUOL algorithm requires $O(B)$ time complexity of computing all

the $DA_{t,j}$'s. After removing the SV with the largest $DA_{t,j}$, the complexity of updating all the values of $y_j f_t(\mathbf{x}_j)$ is $O(B)$. Furthermore, combining the above discussion with the fact that the original DUOL's time complexity is $O(B)$, we can conclude that the overall time complexity of this BDUOL algorithm is also $O(B)$. As for the memory cost, since only B SVs, their weight parameters and the $y_j f_t(\mathbf{x}_j)$s have to be cached, the space complexity is thus also $O(B)$.

3.2 BDUOL Algorithm by Projection Strategy

Although the above removal strategy is optimized to find the best support vector that maximizes the dual ascent (i.e., minimizes the loss of dual ascent caused by budget maintenance), it is still unavoidable to result in the loss of the dual ascent due to the removal of one existing support vector. To minimize such loss, we propose a projection strategy for budget maintenance.

Specifically, in the projection strategy, the selected j-th SV for removal will be projected to the space spanned by the rest SVs. The objective is to find the function closest to $\gamma_j y_j \kappa(\mathbf{x}_j, \cdot)$ in the space spanned by the remaining SVs, or formally:

$$\min_{\beta_i \in [-\gamma_i, C-\gamma_i], i \neq j} \|\gamma_j y_j \kappa(\mathbf{x}_j, \cdot) - \sum_{i \neq j} \beta_i y_i \kappa(\mathbf{x}_i, \cdot)\|^2_{\mathcal{H}_\kappa}. \tag{8}$$

which is essentially a Quadratic Programming (QP) problem. So, we can exploit the existing efficient QP solvers to find the optimal solution.

However, solving the above QP problem directly may not be efficient enough for online learning purpose. To further improve the efficiency, we also proposed an approximate solution by firstly solving the unconstrained optimization problem and then projecting the solution into the feasible region of the constraints. Specifically, setting the gradient of the above equation with respect to $\overline{\beta} = [\beta_i]^\top$, $i \neq j$ as zero, one can obtain the optimal solution as

$$\overline{\beta} = \gamma_j y_j \mathbf{K}^{-1} \mathbf{k}_j ./ \overline{y}, \tag{9}$$

where $\mathbf{K}$ is the kernel matrix for $\mathbf{x}_i, i \neq j$, $\mathbf{k}_j = [\kappa(\mathbf{x}_i, \mathbf{x}_j)]^\top$, $i \neq j$, $./$ is element-wise division and $\overline{y} = [y_i]^\top$, $i \neq j$. In the above, inverting $\mathbf{K}$ can be efficiently realized by using Woodbury formula [12]. As a result, we should set

$$\Delta f_t(\cdot) = \gamma_j y_j \kappa(\mathbf{x}_j, \cdot) - \sum_{i \neq j} \beta_i y_i \kappa(\mathbf{x}_i, \cdot). \tag{10}$$

However, the resultant $f_t - \Delta f_t$'s SV weights may not fall into the range $[0, C]$. To fix this problem, we will project each β_i as follows:

$$\overline{\beta} = \Pi_{[-\overline{\gamma}, C-\overline{\gamma}]}(\gamma_j y_j \mathbf{K}^{-1} \mathbf{k}_j ./ \overline{y}), \tag{11}$$

where $\Pi_{[a,b]}(x) = \max(a, \min(b, x))$ and $\overline{\gamma} = [\gamma_i]^\top$, $i \neq j$.

Now the key problem is to find the best SV among $B+1$ candidates for projection. Since after the projection of $\gamma_j y_j \kappa(\mathbf{x}_j, \cdot)$, the resultant dual ascent is

$$DA_{t,j} = -\sum_i \Delta\gamma_i + \sum_i \Delta\gamma_i y_i f_t(\mathbf{x}_i) - \frac{1}{2}\|\Delta f_t\|^2_{\mathcal{H}_\kappa}. \tag{12}$$

So the best SV is the one which can achieve the largest value $DA_{t,j}$.

Complexity Analysis. The time complexity for BDUOL is dominated by the computation K^{-1}. Computing K^{-1} using K will cost $O(B^3)$ time, however using Woodbury formula, we can efficiently compute it using only $O(B^2)$ time. In addition, we need to compute B times projections for every SV, so the total time complexity for one step of updating is $O(B^3)$. Finally, the memory burden for the BDUOL algorithm is $O(B^2)$, since the storage of kernel matrix K and inverse kernel matrix K^{-1} dominated the main memory cost.

3.3 BDUOL Algorithm by Nearest Neighbor Strategy

The above projection strategy is able to achieve a better improvement of dual ascent than the removal strategy, it is however much more computationally expensive. To balance the tradeoff between efficiency and effectiveness, we propose an efficient nearest neighbor strategy which approximates the projection strategy by projecting the removed SV to its nearest neighbor SV, based on the distance in the mapped feature space. This strategy is motivated by the fact that the nearest neighbor SV usually could be a good representative of the removed SV. Using this strategy, we can significantly improve the time efficiency. In particular, as we use only the nearest neighbor for projection, the corresponding solution according to equation 11 can be expressed:

$$\beta_{N_j} = \Pi_{[-\gamma_{N_j}, C-\gamma_{N_j}]}(\gamma_j y_j \kappa(\mathbf{x}_{N_j}, \mathbf{x}_{N_j})^{-1} \kappa(\mathbf{x}_{N_j}, \mathbf{x}_j)/y_{N_j}), \tag{13}$$

where $\kappa(\mathbf{x}_{N_j}, \cdot)$ is the nearest neighbor of $\kappa(\mathbf{x}_j, \cdot)$. As a result, the corresponding $\Delta f_t = \gamma_j y_j \kappa(\mathbf{x}_j, \cdot) - \beta_{N_j} y_{N_j} \kappa(\mathbf{x}_{N_j}, \cdot)$. Since after the projection of $\gamma_j y_j \kappa(\mathbf{x}_j, \cdot)$, the resultant dual ascent is

$$DA_{t,j} = -\sum_i \Delta\gamma_i + \sum_i \Delta\gamma_i y_i f_t(\mathbf{x}_i) - \frac{1}{2}\|\Delta f_t\|^2_{\mathcal{H}_\kappa}. \tag{14}$$

Thus, the best SV is the one that has the largest value $DA_{t,j}$.

Complexity Analysis. All the computation steps except looking for the nearest neighbor have constant time complexity of $O(1)$. For improving nearest neighbor searching, we can cache the indexes and the distances about the nearest neighbors of the SVs, making finding the nearest neighbor in $O(1)$. After remove the best SV, we should also update the caches. The time complexity for this updating is at most $O(B)$. In summary, the time complexity for the overall strategy is $O(B)$, and the overall memory cost is also $O(B)$.

Remark: To improve the efficiencies of the projection and nearest neighbor strategies, we could use the projection or the nearest neighbor strategy to keep the information from the SV selected by the removal strategy and then remove it, since the search cost of the removal strategy is quite lower than the other two strategies.

4 Experimental Results

In this section, we evaluate the empirical performance of the proposed algorithms for Budget Double Updating Online Learning (BDUOL) by comparing them with the state-of-the-art algorithms for budget online learning.

4.1 Algorithms for Comparison

In our experiments, we implement the proposed BDUOL algorithms as follows:

- "BDUOL$_{remo}$": the BDUOL algorithm by the **remo**val strategy for budget maintenance described in section 3.1,
- "BDUOL$_{proj}$": the BDUOL algorithm by the exact **proj**ection strategy by a standard QP solver for budget maintenance described in section 3.2,
- "BDUOL$_{appr}$": the BDUOL algorithm by the **appr**oximate projection strategy for budget maintenance described in section 3.2,
- "BDUOL$_{near}$": the BDUOL algorithm by the **near**est neighbor strategy for budget maintenance described in section 3.3,

For comparison, we include the following state-of-the-art algorithms for budget online learning:

- "RBP": the Random Budget Perceptron algorithm [9],
- "Forgetron": the Forgetron algorithm [8],
- "Projectron": the Projectron algorithm [10], and
- "Projectron++": the aggressive version of Projectron algorithm [10].

Besides, we also include two non-budget online learning algorithms as yardstick:

- "Perceptron": the classical Perceptron algorithm [6], and
- "DUOL": the Double Updating Online Learning algorithm [4].

4.2 Experimental Testbed and Setup

Table 1. Details of the datasets in our experiments.

Dataset	# instances	# features
german	1000	24
MITface	6977	361
mushrooms	8124	112
spambase	4601	57
splice	3175	60
w7a	24692	300

We test all the algorithms on six benchmark data sets from web machine learning repositories listed in Table 1. These data sets can be downloaded from LIBSVM website[1], UCI machine learning repository[2], and MIT CBCL face

[1] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.
[2] `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

data sets[3]. These datasets were chosen fairly randomly to cover various sizes of datasets.

To make a fair comparison, all the algorithms in our comparison adopt the same experimental setup. A gaussian kernel is adopted in our study, for which the kernel width is set to 8 for all the algorithms and datasets. To make the number of support vectors fixed for Projectron and Projectron++ algorithms, we simply store the received SVs before the budget overflows, and then project the subsequent ones into the space spanned by the stored SVs afterward. The penalty parameter C in the DUOL algorithm was selected by 5-fold cross validation for all the datasets from range $2^{[-10:10]}$. Due to the cross-validation, we randomly divide every dataset into two equal subsets: cross validation (CV) dataset and online learning dataset. And C is set as the same value with DUOL. Furthermore, the value ρ for the DUOL and its budget variants is set as 0, according to the previous study on the effect of ρ.

The budget sizes B for different datasets are set as proper fractions of the support vector size of Perceptron, which are shown in Table 3. All the experiments were conducted 20 times, each with a different random permutation of data points. All the results were reported by averaging over the 20 runs. For performance metrics, we evaluate the online classification performance by evaluating online cumulative mistake rates and running time cost.

4.3 Performance Evaluation of Non-budget Algorithms

Table 2 summarizes the average performance of the two non-budget algorithms for kernel-based online learning. First of all, similar to the previous study [4], we found that DUOL outperforms Perceptron significantly for all the datasets according to t-test results, which validates our motivation of choosing DUOL as the basic online learning algorithm for budget online learning. Second, we noticed that the support vector size of DUOL is in general much larger than that of Perceptron. Finally, the time cost of DUOL is much higher than that of Perceptron, mostly due to the larger number of support vectors. Both the large number of support vectors and high computational time motivate the need of studying budget DUOL algorithms in this work.

Algorithm	Perceptron			DUOL		
Datasets	Mistake (%)	Support Vectors (#)	Time (s)	Mistakes (%)	Support Vectors (#)	Time (s)
german	35.760 %± 1.149	178.800 ± 5.745	0.007	29.820 %± 1.243	381.750 ± 5.866	0.044
MITface	6.246 %± 0.252	217.85 ± 8.774	0.04	2.418 %± 0.156	408.5 ± 9.339	0.114
mushrooms	3.175 %± 0.463	128.950 ± 18.805	0.040	0.591 %± 0.086	247.050 ± 14.084	0.099
spambase	27.354 %± 0.561	629.150 ± 12.906	0.050	22.680 %± 0.557	1385.800 ± 17.519	0.317
splice	21.808 %± 0.709	346.100 ± 11.257	0.026	15.633 %± 0.461	777.450 ± 11.551	0.130
w7a	4.366 %± 0.093	539.000 ± 11.530	0.272	3.068 %± 0.114	1171.600 ± 30.396	0.728

Table 2. Evaluation of non-budget algorithms on the the data sets.

[3] `http://cbcl.mit.edu/software-datasets`.

4.4 Performance Evaluation of Budget Algorithms

Table 3 summarizes the results of different budget online learning algorithms. We can draw several observations.

First of all, we observe that RBP and Forgetron achieve very similar performance for most cases. In addition, we also find that Projectron++ achieves a lower mistake rate than Projectron for almost all the datasets and for varied budget sizes, which is similar to the previous results reported in [10]. Moreover, compared with the baseline algorithms RBP and Forgetron, the proposed BDUOL$_{remo}$ algorithm using a simple removal strategy achieves comparable or better mistake rate when the budget size is large, but fails to improve when the budget size is very small, which indicates a simple removal strategy may not be always effective and a better budget maintenance strategy is needed.

Second, among all the algorithms in comparison for budget online learning, we find that BDUOL$_{proj}$ always achieves the lowest mistake rates for most cases. These promising results indicate the projection strategy can effectively reduce the information loss. However, we also notice that the time cost of the BDUOL$_{proj}$ is among the highest ones, which indicates it is important to find some more efficient strategy.

Third, by comparing two approximate strategies, we find that BDUOL$_{appr}$ achieves better mistake rates than BDUOL$_{near}$ only on the german and mushrooms datasets, while it consumes too much time than the proposed BDUOL$_{near}$ algorithms, which indicates BDUOL$_{near}$ achieves better trade off between mistake rates and time complexity than BDUOL$_{appr}$. In addition, when the number of budget is large, BDUOL$_{near}$ always achieves similar performance with the BDUOL$_{proj}$, while consumes significantly less time, which indicates the proposed nearest neighbor strategy is a good alternative of the projection strategy.

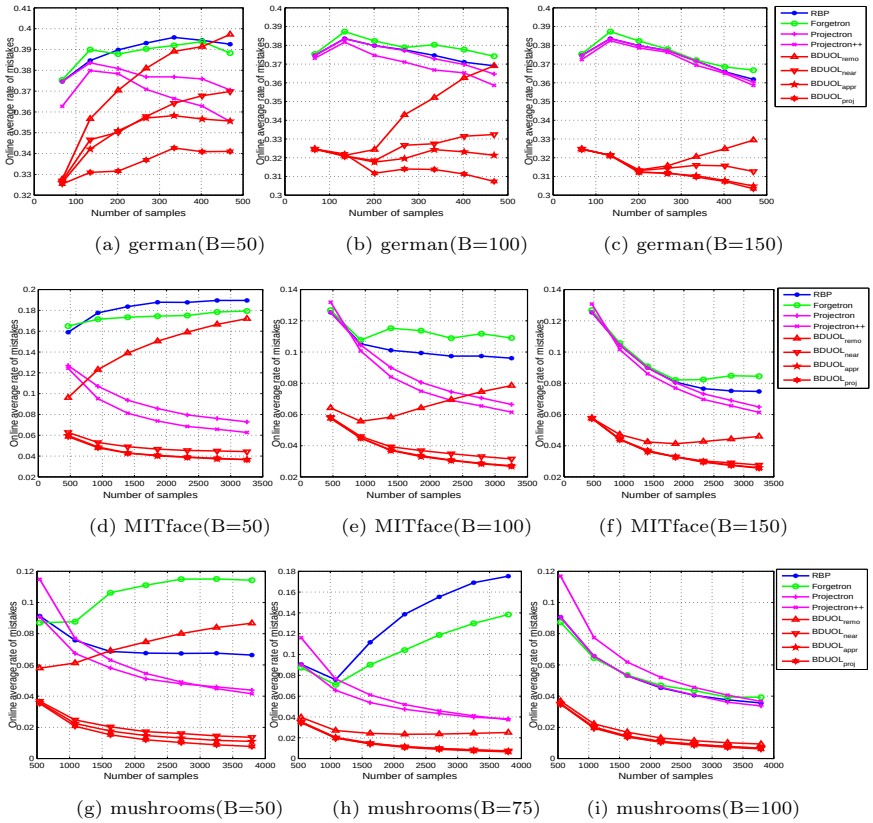
Finally, Figure 2 and Figure 3 show the detailed online evaluation processes of the several budget online learning algorithms. Similar observations from these figures further verified the efficacy of the proposed BDUOL technique.

5 Conclusions

This paper presented a new framework of budget double updating online learning for kernel-based online learning on a fixed budget, which requires the number of support vectors associated with the prediction function is always bounded by a predefined budget. We theoretically analyzed its performance, which reveals that its effectiveness is tightly connected with the dual ascent achieved by the model reduction for budget maintenance. Based on the theoretical analysis, we proposed three budget maintenance strategies: removal, projection, and nearest neighbor. We evaluate the proposed algorithms extensively on benchmark datasets. The promising empirical results show that the proposed algorithms outperform the state-of-the-art budget online learning algorithms in terms of mistake rates. Future work will exploit different budget maintenance strategies and extend the proposed work to multi-class budgeted online learning.

Budget Size		B=50		B=100		B=150	
Dataset	Algorithm	Mistake (%)	Time (s)	Mistakes (%)	Time (s)	Mistakes (%)	Time (s)
german	RBP	39.140 %± 1.338	0.010	36.940 %± 1.837	0.009	36.080 %± 1.392	0.008
	Fogetron	38.840 %± 1.551	0.014	37.250 %± 1.399	0.013	36.570 %± 1.641	0.013
	Projectron	36.990 %± 1.652	0.020	36.310 %± 1.441	0.027	35.870 %± 1.149	0.063
	Projectron++	35.370 %± 1.413	0.036	35.620 %± 1.251	0.046	35.680 %± 1.380	0.103
	BDUOL_{remo}	39.970 %± 3.150	0.068	37.180 %± 2.297	0.077	33.330 %± 2.264	0.088
	BDUOL_{near}	37.090 %± 1.763	0.098	33.330 %± 1.697	0.112	31.360 %± 1.511	0.129
	BDUOL_{appr}	35.540 %± 2.010	0.160	32.340 %± 1.570	0.381	30.450 %± 1.338	0.941
	BDUOL_{proj}	**34.030** %± 1.104	0.214	**30.710** %± 1.261	0.624	**30.330** %± 1.221	1.341
Budget Size		B=50		B=100		B=150	
Dataset	Algorithm	Mistake (%)	Time (s)	Mistakes (%)	Time (s)	Mistakes (%)	Time (s)
MITface	RBP	18.964 %± 1.330	0.063	9.557 %± 0.615	0.053	7.463 %± 0.609	0.050
	Fogetron	18.038 %± 1.470	0.081	10.707 %± 0.883	0.077	8.373 %± 0.546	0.076
	Projectron	7.137 %± 0.384	0.086	6.461 %± 0.288	0.098	6.316 %± 0.304	0.142
	Projectron++	6.135 %± 0.312	0.149	5.973 %± 0.215	0.197	5.978 %± 0.219	0.373
	BDUOL_{remo}	17.516 %± 2.264	0.225	8.096 %± 0.811	0.208	4.700 %± 0.449	0.194
	BDUOL_{near}	4.435 %± 0.396	0.179	3.078 %± 0.304	0.187	2.701 %± 0.277	0.205
	BDUOL_{appr}	3.632 %± 0.277	0.249	**2.618** %± 0.221	0.429	2.501 %± 0.258	0.836
	BDUOL_{proj}	**3.611** %± 0.319	0.273	2.645 %± 0.228	0.492	**2.481** %± 0.168	0.755
Budget Size		B=50		B=75		B=100	
Dataset	Algorithm	Mistake (%)	Time (s)	Mistakes (%)	Time (s)	Mistakes (%)	Time (s)
mushrooms	RBP	6.551 %± 1.056	0.054	17.841 %± 1.539	0.073	3.488 %± 0.713	0.049
	Fogetron	11.273 %± 1.750	0.085	14.154 %± 2.748	0.094	3.895 %± 1.141	0.079
	Projectron	4.264 %± 0.613	0.095	3.703 %± 0.731	0.099	3.207 %± 0.473	0.107
	Projectron++	3.986 %± 0.297	0.161	3.557 %± 0.174	0.178	3.484 %± 0.117	0.197
	BDUOL_{remo}	8.754 %± 2.438	0.210	2.547 %± 0.865	0.181	0.891 %± 0.152	0.162
	BDUOL_{near}	1.329 %± 0.198	0.141	0.710 %± 0.095	0.132	0.618 %± 0.081	0.131
	BDUOL_{appr}	1.065 %± 0.218	0.193	0.667 %± 0.092	0.194	0.623 %± 0.080	0.248
	BDUOL_{proj}	**0.729** %± 0.060	0.230	**0.604** %± 0.080	0.266	**0.577** %± 0.071	0.290
Budget Size		B=200		B=400		B=600	
Dataset	Algorithm	Mistake (%)	Time (s)	Mistakes (%)	Time (s)	Mistakes (%)	Time (s)
spambase	RBP	31.826 %± 0.924	0.065	29.220 %± 0.550	0.069	27.417 %± 0.598	0.059
	Fogetron	32.461 %± 0.971	0.077	29.641 %± 0.742	0.084	27.424 %± 0.644	0.082
	Projectron	29.237 %± 0.750	0.238	27.480 %± 0.484	0.929	27.750 %± 2.134	2.776
	Projectron++	28.822 %± 0.725	0.819	28.693 %± 6.781	3.874	27.559 %± 1.572	7.277
	BDUOL_{remo}	34.559 %± 1.308	0.522	28.989 %± 0.927	2.013	25.661 %± 0.709	4.541
	BDUOL_{near}	28.180 %± 1.084	0.756	25.607 %± 0.748	3.874	24.187 %± 0.584	8.570
	BDUOL_{appr}	28.950 %± 2.001	5.555	26.843 %± 0.948	16.396	24.846 %± 0.695	30.186
	BDUOL_{proj}	**27.448** %± 0.961	8.837	**25.307** %± 0.793	53.987	**23.780** %± 0.640	197.199
Budget Size		B=100		B=200		B=300	
Dataset	Algorithm	Mistake (%)	Time (s)	Mistakes (%)	Time (s)	Mistakes (%)	Time (s)
splice	RBP	30.003 %± 1.318	0.035	25.126 %± 0.900	0.033	22.530 %± 0.967	0.028
	Fogetron	29.912 %± 1.483	0.044	25.545 %± 0.582	0.043	22.457 %± 0.734	0.041
	Projectron	22.851 %± 0.756	0.061	22.007 %± 1.023	0.146	21.830 %± 0.811	0.356
	Projectron++	22.628 %± 0.649	0.163	21.843 %± 1.002	0.445	21.919 %± 1.188	0.922
	BDUOL_{remo}	26.150 %± 1.279	0.238	22.117 %± 1.321	0.409	18.299 %± 0.797	0.641
	BDUOL_{near}	23.125 %± 1.161	0.435	18.847 %± 0.866	0.458	16.991 %± 0.600	1.201
	BDUOL_{appr}	**19.779** %± 0.754	1.110	18.062 %± 2.000	2.596	17.927 %± 2.065	3.905
	BDUOL_{proj}	20.243 %± 0.667	1.041	**17.190** %± 0.734	2.685	**16.191** %± 0.517	5.475
Budget Size		B=300		B=400		B=500	
Dataset	Algorithm	Mistake (%)	Time (s)	Mistakes (%)	Time (s)	Mistakes (%)	Time (s)
w7a	RBP	4.681 %± 0.257	0.291	4.597 %± 0.227	0.316	4.434 %± 0.108	0.332
	Fogetron	4.697 %± 0.102	0.367	4.584 %± 0.180	0.390	4.466 %± 0.137	0.398
	Projectron	4.680 %± 0.290	0.727	4.625 %± 0.456	1.174	4.406 %± 0.130	1.929
	Projectron++	3.880 %± 0.526	4.366	3.672 %± 0.214	7.763	3.666 %± 0.152	8.796
	BDUOL_{remo}	3.832 %± 0.170	0.995	3.361 %± 0.145	1.824	**3.175** %± 0.133	2.401
	BDUOL_{near}	3.572 %± 0.114	1.241	3.269 %± 0.133	3.324	3.183 %± 0.087	4.859
	BDUOL_{appr}	4.126 %± 0.502	4.553	4.001 %± 0.153	6.794	3.775 %± 0.121	9.646
	BDUOL_{proj}	**3.367** %± 0.096	11.306	**3.227** %± 0.126	18.965	3.254 %± 0.332	29.807

Table 3. Evaluation of several budgeted algorithms with varied budget sizes.



(a) german(B=50) (b) german(B=100) (c) german(B=150)

(d) MITface(B=50) (e) MITface(B=100) (f) MITface(B=150)

(g) mushrooms(B=50) (h) mushrooms(B=75) (i) mushrooms(B=100)

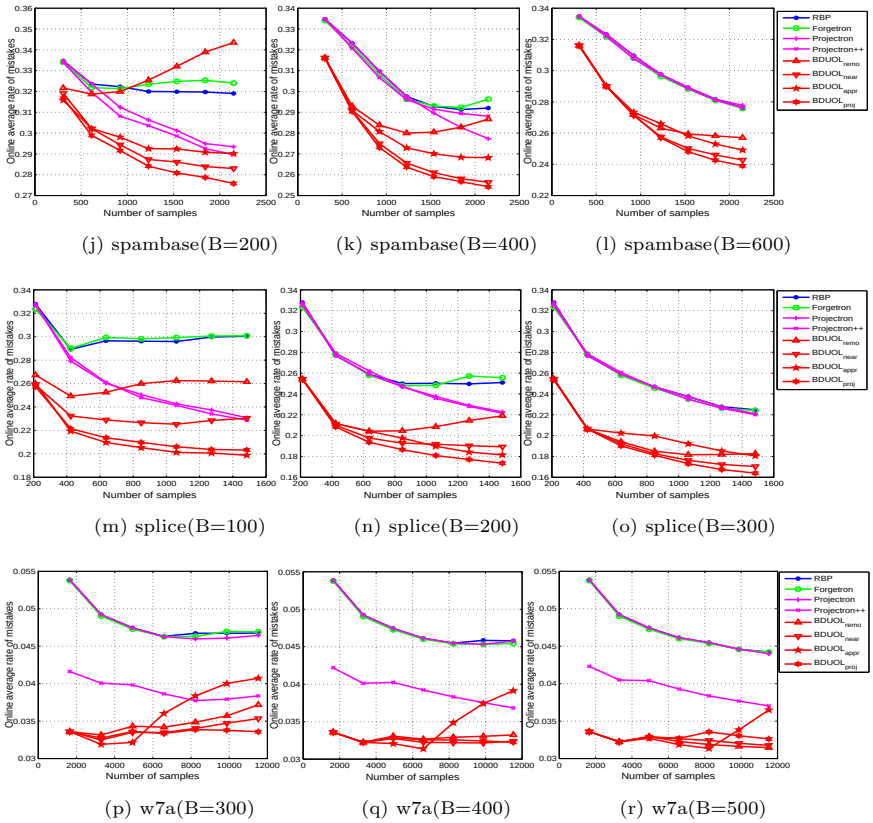
Fig. 2. Evaluation of online mistake rates against the number of samples on three datasets. The plotted curves are averaged over 20 random permutations.

Acknowledgments

This work was supported by Singapore MOE tier 1 grant (RG33/11) and Microsoft Research grant (M4060936).

References

1. Jyrki Kivinen, Alexander J. Smola, and Robert C. Williamson. Online learning with kernels. *IEEE Transactions on Signal Processing*, 52(8):2165–2176, 2004.
2. Li Cheng, S. V. N. Vishwanathan, Dale Schuurmans, Shaojun Wang, and Terry Caelli. Implicit online learning with kernels. In *NIPS*, pages 249–256, 2006.
3. Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.
4. Peilin Zhao, Steven C. H. Hoi, and Rong Jin. Double updating online learning. *Journal of Machine Learning Research*, 12:1587–1615, 2011.



(j) spambase(B=200) (k) spambase(B=400) (l) spambase(B=600)

(m) splice(B=100) (n) splice(B=200) (o) splice(B=300)

(p) w7a(B=300) (q) w7a(B=400) (r) w7a(B=500)

Fig. 3. Evaluation of online mistake rates against the number of samples on three datasets. The plotted curves are averaged over 20 random permutations.

5. Koby Crammer, Jaz S. Kandola, and Yoram Singer. Online classification on a budget. In *NIPS*, 2003.
6. Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–407, 1958.
7. Jason Weston and Antoine Bordes. Online (and offline) on an even tighter budget. In *AISTATS*, pages 413–420, 2005.
8. Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. The forgetron: A kernel-based perceptron on a budget. *SIAM J. Comput.*, 37(5):1342–1372, 2008.
9. Giovanni Cavallanti, Nicolò Cesa-Bianchi, and Claudio Gentile. Tracking the best hyperplane with a simple budget perceptron. *Machine Learning*, 69(2-3):143–167, 2007.
10. Francesco Orabona, Joseph Keshet, and Barbara Caputo. Bounded kernel-based online learning. *Journal of Machine Learning Research*, 10:2643–2666, 2009.
11. Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.
12. Gert Cauwenberghs and Tomaso Poggio. Incremental and decremental support vector machine learning. In *Advances in Neural Information Processing Systems 13 (NIPS)*, pages 409–415, 2000.