Proceedings of International Joint Conference on Neural Networks, Atlanta, Georgia, USA, June 14-19, 2009

Tighter Perceptron with Improved Dual Use of Cached Data for Model Representation and Validation

Zhuang Wang and Slobodan Vucetic

Abstract—Kernel Perceptrons are represented by a subset of training points, called the support vectors, and their associated weights. To address the issue of unlimited growth in model size during training, budget kernel perceptrons maintain the fixed number of support vectors and thus achieve the constant update time and space complexity. In this paper, a new kernel perceptron algorithm for online learning on a budget is proposed. Following the idea of Tighter Perceptron, upon exceeding the budget, the algorithm removes the support vector with the minimal impact on classification accuracy. To optimize memory use, instead on maintaining a separate validation data set for accuracy estimation, the proposed algorithm only uses the support vectors for both model representation and validation. This is achieved by estimating posterior class probability of each support vector and using this information in validation. The experimental results on 11 benchmark data sets indicate that the proposed algorithm is significantly more accurate than the competing budget kernel perceptrons and that it has comparable accuracy to the resource unbounded perceptrons, including the original kernel perceptron and the Tighter Perceptron that uses whole training data set for validation.

I. Introduction

THE invention of the Support Vector Machines [12] attracted a lot of interest in adapting the kernel methods for both batch and online learning. Kernel perceptrons [5, 7, 8, 9] are a popular class of algorithms for online learning. They are represented by a subset of observed examples, called the support vectors, and their weights. The baseline kernel perceptron algorithm is simple – it observes a new example and, if it is misclassified by the current model, adds it to the model as a new support vector. The popularity of kernel perceptrons is due to their ease of implementation, the ability to achieve quite competitive classification accuracy to the batch mode alternatives, and the existence of theoretical results characterizing their behavior.

In addition to their appealing properties, kernel perceptrons often suffer from an unbounded growth in the number of support vectors with training data size. This, in turn, causes unbounded growth in training time and space needed to store the classifier. Such behavior is unacceptable in many practical online learning applications. To address the issue of unbounded growth in computational resources, a class of online kernel perceptron algorithms on a fixed budget has been developed. To maintain the budget, the proposed algorithms typically decide to discard one of the support vectors when the budget is exceeded upon addition of a new support vector. While there are theoretical guarantees for convergence of several budget kernel perceptron algorithms, their actual performance is often poor on noisy classification problems.

Among budget kernel perceptrons, the Tighter Perceptron algorithm [13] is one of the most successful in practice. It removes the support vector that has the minimal positive impact on the classification accuracy. To estimate the accuracy, the algorithm requires maintenance of an additional validation data set. When the validation set is large, Tighter Perceptron is able to achieve respectable classification accuracy. However, the existence of validation set also puts an increasing burden on training speed and memory. When, in order to decrease the budget, the support vector set is used both for model representation and validation, the accuracy decreases dramatically. The main reason for such behavior is that support vectors are a biased sample of training examples that were incorrectly classified. Therefore, support vectors are likely to contain an overwhelming amount of noisy training examples and could therefore provide quite misleading accuracy estimates.

In this paper, we propose a new algorithm, called here for convenience the Tightest Perceptron, that manages to obtain highly accurate estimates of classification accuracy using exclusively the support vectors. To achieve this, a simple data summary is maintained along each support vector to estimate the posterior class probability for each support vector. During the validation, the expectation of the validation hinge loss is calculated based on the estimated class probabilities. The experimental results show that the proposed algorithm has impressive performance on 11 benchmark data sets.

II. Problem Setting and Previous Work

We study the online learning for binary classification. Online learning is performed in a sequence of consecutive rounds. On each round, the algorithm observes an example from the training set. An example is a pair (x, y), where x is an M-dimensional attribute vector and $y \in \{+1, -1\}$ is the associated binary label. The independent and identically distributed training set D is a sequence of examples $(x_1, y_1),\dots,(x_N, y_N)$ and can only be observed in a single pass.

The classical perceptron algorithm [11] has the following training procedure. Initially, the prediction model $f(x)$ is set to zero, $f(x) = 0$. In round t, the new example (x_t, y_t) is observed

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

Zhuang Wang and Slobodan Vucetic are with the Center for Information Science and Technology, Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA, email: zhuang@temple.edu, vucetic@ist.temple.edu.

and its label is predicted by the current model $f(x)$ as $sign(f(x_t))$. If the margin of this example, defined as the product $y_t \cdot f(x_t)$, is below threshold $\beta = 0$ (i.e. $y_t f(x_t) \le \beta$), then weight $\alpha_t = 1$ is assigned to this example and the model is updated as $f(x) = f(x) + \alpha_t y_t x_t \cdot x$. Each example added to the model is called the Support Vector (SV). If the example is correctly classified, its weight is set to $\alpha_t = 0$, and the model is thus not updated.

Alternatively, the algorithm can also modify the current hypothesis by multiplying it with scalar ϕ_t ($f_t(x) = \phi_t f_t(x)$). The standard parameter values of $\beta = 0$, $\alpha_t = 1$, and $\phi_t = 1$ can be chosen differently, as was done in ALMA [7], ROMMA [9], NORMA [8] and PA [5] algorithms. In this paper, we will consider kernel perceptrons with the standard parameter values.

The classical perceptron implies a linear decision function. It could be made nonlinear by using $\Phi(x)$ as attributes instead of x, where Φ is a nonlinear mapping of the original attribute space into the feature space. If there exists a kernel function k such that $\Phi(x_i) \cdot \Phi(x) = k(x_i, x)$ [1], the model $f(x)$ can be represented as

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i \Phi(x_i) \cdot \Phi(x) = \sum_{i=1}^{N} \alpha_i y_i k(x_i, x)$$

and is denoted as *the kernel perceptron*. It is important to note that the kernel function k allows us to express the model in terms of the original attributes and avoid explicitly working in the potentially high (or infinite) dimensional feature space.

A. Budget Perceptron Algorithms

In spite of the powerful performance, kernel methods often suffer from an unbounded growth in the number of support vectors with training data. This creates serious problems in both training and testing phase because the time needed to compute $f(x)$ and the space needed to store the model scales linearly with the number of SVs. In many practical online applications where a short feedback time and bounded space is a requirement, the unbounded growth mentioned above is not acceptable. This fact motivated work in developing online algorithms on a fixed budget.

1) Fixed Budget Perceptron: The pioneering work was done in [4] to address the problem. There, a standard kernel perceptron was modified by adding a support vector removal procedure to keep the budget. Let us denote I_t as the set of support vectors at round t of the kernel perceptron algorithm. If the number of support vectors exceeds the predefined budget B in round t (i.e. $|I_t|>B$), support vector with the largest margin,

$$\arg\max_{j \in I_t} \left\{ y_j \left(f(x_j) - \alpha_j y_j k(x_j, x_j) \right) \right\},$$

is removed. While this algorithm achieves respectable accuracy 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 the noisy examples, resulting in degradation of accuracy.

2) Random Perceptron: The simplest removal procedure is to remove a randomly selected support vector. Despite its simplicity, this algorithm often has satisfactory performance. In addition, the algorithm's convergence has also been proven [2].

3) Forgetron: A more advanced removal procedure was developed in [6] by introducing a forgetting factor. After each update step, forgetting factor $0 < \phi_t < 1$ is used to scale the current model (and all its support vectors). The oldest support vector (with the smallest weight) is removed if budget is exceeded. The algorithm's convergence has also been proven.

4) Tighter Perceptron: [13] proposed to remove the support vector that has the smallest positive influence on accuracy. To allow accuracy estimation, an additional validation set composed of the previously observed training examples is maintained. Specifically, on the t-th round where $|I_t|>B$, the algorithm removes j-th support vector with

$$\arg\min_{j \in I_t} \left\{ \sum_{k \in V_t} l_{0-1}\left(y_k, \operatorname{sign}\left(f(x_k) - \alpha_j y_j k(x_j, x_k) \right) \right) \right\},$$

where V_t is the validation set and $l_{0\text{-}1}$ denotes the 0-1 classification loss.

From the perspective of accuracy estimation, it is ideal to use all the previously seen training examples for validation. However, the use of validation set puts an additional burden to the memory budget and the training time. Due to practical considerations, the size of validation data set should be restricted. There are several variants of the Tighter algorithm depending on the size of the validation set: Tighter^{Full} uses all training examples for validation, Tighter^{A} uses selected A examples that are disjoint from the support vectors, and Tighter^{0} uses support vectors. While Tighter^{A} and Tighter^{0} are budget algorithms, their accuracy estimates are less reliable. That is especially the case for Tighter^{0} because the support vector set is a biased sample from training data that is likely to contain disproportionally large fraction of noisy examples.

In the following section, a statistically-based method is proposed to improve accuracy estimation using only the support vector set.

III. The Proposed Algorithm

The main property of the proposed algorithm is an improved accuracy validation using only the support vector set. The validation improvement is possible when posterior class probabilities of support vectors are used instead of their actual labels (as is done in Tighter^{0}). The open problem with this approach is that posterior class probabilities of support vectors are unknown and should be estimated. Our idea is that a high-quality class probability estimate for each support vector can be obtained by looking at labels of training examples in its neighborhood. We call the resulting algorithm the Tightest Perceptron or Tightest, in short.

The proposed algorithm is sketched in Figure 1. Instead of simply discarding the selected support vector or the new training point that does not become the support vector, we are using its class information to improve the class probability estimate of its nearest support vector. To implement the idea, the i-th SV is represented by tuple (x_i, y_i, c_i^+, c_i^-) containing its

```
Input: $(x_1, y_1),...,(x_N, y_N)$, budget $B$
Initialization: $f(x) = 0$, $S = \varnothing$
Output: $f(x)$

for $i$=1 to $N$
    if $y_i f(x_i) \le 0$
        $f(x) = f(x) + y_i k(x_i,x)$
        if $y_i=1$
            $S = S \cup \{(x_i, y_i, 1, 0)\}$
        else $S = S \cup \{(x_i, y_i, 0, 1)\}$
        if $|S| > B$
            $r = \arg\min_{j \in S} loss(f(x) - y_j k(x_j, x))$
            $f(x) = f(x) - y_r k(x_r,x)$
            $S = S - \{(x_r, y_r, c_r^+, c_r^-)\}$
            $UpdateSummary(S, x_r, c_r^+, c_r^-)$
    else
        if $y_i=1$
            $UpdateSummary(S, x_i, 1, 0)$
        else $UpdateSummary(S, x_i, 0, 1)$
```

```
Subroutine $UpdateSummary(S, x, c^+, c^-)$
    $k = \arg\min_{j \in S} \| x_j - x \|$
    $c_k^+ = c_k^+ + c^+ \cdot k(x, x_k)$
    $c_k^- = c_k^- + c^- \cdot k(x, x_k)$
```

Fig 1. The pseudo code for Tightest

attribute values x_i, the original label y_i, and counts c_i^+ and c_i^- that represent the total number of positive and negative training examples observed in its close neighborhood. As will be descried in III.A, these counts are used to estimate the posterior class distribution at x_i. We denote the set of support vectors augmented by the counts with S.

After initializing $f(x)$ to zero and setting the augmented support set S to empty, examples from the training data are read sequentially. If the observed example is well classified, the current model is retained. Before discarding the example, *UpdateSummary* subroutine is used to update count of its nearest support vector. Instead of incrementing the count by one, we use the soft increment that is a function of kernel distance. In this way, larger weight is given to the labels of training examples closest to the support vectors.

If a training example is misclassified, it is added to the current model, and S is updated accordingly. When the number of support vectors exceeds the budget B, $|S| > B$, the algorithm evaluates removal of each SV, selects the one whose removal introduces the least validation loss, and updates the model by removing it. Details of the selection are given in III.A. Before discarding the support vector, its counts are used to update the counts of its nearest support vector.

A. Accuracy Estimation

Given the support vector set S, the best support vector for removal is determined as the one with index

$$r = \arg\min_{j \in S} loss(f(x) - y_j k(x_j, x)),$$

where *loss* is defined as the expected accuracy loss on the support vector set,

$$loss(f(x)) = \frac{1}{|S|} \sum_{i \in S} (p_i^+ l_i^+ + p_i^- l_i^-), \tag{1}$$

where $p_i^+ = P(y_i = 1|x_i)$ and $p_i^- = P(y_i = -1|x_i)$ are the posterior probabilities that x_i is labeled as positive and negative, respectively. Quantity l_i^+ (or l_i^-) denotes the accuracy loss at x_i assuming its class label is actually positive (or negative).

One choice of accuracy loss is the traditional 0-1 loss defined as $l_i^+ = 1$ if $f(x_i) < 0$, and $l_i^+ = 0$ otherwise. A slight problem with 0-1 loss is that it could not distinguish between large and small errors, which can be important when validation data size is small. The alternative choice, implemented in our algorithm, is to use the hinge loss defined as $l_i^+ = max(0,1-(+1)\cdot f(x_i))$ and $l_i^- = max(0,1-(-1)\cdot f(x_i))$.

We observe that, using the introduced notation, in Tighter^0 algorithm $p_i^+ = 1$ and $p_i^- = 0$ if $y_i = +1$, and $p_i^+ = 0$ and $p_i^- = 1$ if $y_i = -1$, and that 0-1 loss is used for l_i^+ and l_i^-.

The remaining issue is estimating value of p_i^+ (observe that $p_i^- = 1 - p_i^+$) based on counts c_i^+ and c_i^- maintained by the algorithm. The maximum likelihood estimate $p_i^+ = c_i^+/(c_i^+ + c_i^-)$ is unreliable when counts are small. Instead, we use the Bayesian approach where p_i^+ is treated as a random variable whose prior has Beta distribution $Beta(a^+,a^-)$, where a^+ and a^- are some positive values (typically set to 1). In this case, the posterior distribution of p_i^+ has Beta distribution $Beta(c_i^+ + a^+, c_i^- + a^-)$. Since we are treating p_i^+ and p_i^- as random variables, we need to modify the accuracy loss in equation (1) to

$$loss(f(x)) = \frac{1}{|S|} \sum_{i \in S} (w_i^+ l_i^+ + (1 - w_i^+) l_i^-), \tag{2}$$

where w_i^+ is calculated as

$$w_i^+ = \int_{0.5}^{1} Beta(x \,|\, c_i^+ + a^+, c_i^- + a^-) dx.$$

B. Complexity

The space requirement of the proposed Tightest perceptron is constant in training size and scales as $O(B)$ with the budget B, because only B support vectors are maintained in the memory. Let us now consider the time complexity. The prediction for the new coming example takes $O(B)$ runtime. With some bookkeeping (predictions of the current perceptron on each support vector should be maintained), the evaluation of accuracy loss after removal of a single SV requires $O(B)$ time, and there are B+1 such evaluations. Finding the nearest neighbor in *UpdateSummary* subroutine costs another $O(B)$. Therefore, the total runtime for an update is $O(B^2)$ and the total training time for a data set of size N is $O(NB^2)$.

IV. Experiments

In this section, we present results of detailed evaluation of the proposed Tightest perceptron on a number of benchmark datasets.

A. Data sets

Properties of 11 benchmark data sets for binary classification are summarized in Table 1. The multi-class data sets were converted to two-class sets as follows. For the digit datasets *Pendigits* and *USPS* we converted the original 10-class problems to binary by representing digits 1, 2, 4, 5, 7 (non-round digits) as negative class and digits 3, 6, 8, 9, 0 (round digits) as positive class. For *Letter* dataset, negative class was created from the first 13 letters of the alphabet and positive class from the remaining 13. The 10-class *MNIST* data set was simplified to binary data by separating digit 3 from digit 8. Class 1 in the 3-class *Waveform* was treated as negative and the remaining two as positive. For *Covertype* data the class 2 was treated as positive and the remaining 6 classes as negative. *Adult9*, *Banana*, *Gauss*, and *IJCNN* were originally 2-class data sets. *NCheckerboard* data was generated as a uniformly distributed two-dimensional 4 × 4 checkerboard with alternating class assignments where class

TABLE 1
DATA SET AND KERNEL PARAMETER SUMMARIES

Data sets	Training	Testing	Dim	δ^2
Adult9	30162	15060	123	25
Banana	4300	1000	2	0.1
NCheckerboard	10000	5000	2	0.1
Covertype	100000	100000	54	54/2
Gauss	10000	5000	2	0.1
IJCNN	49990	91701	22	22/2
Letter	16000	4000	16	1
MNIST	11982	1984	784	784/2
Pendigits	7494	3498	16	16/2
USPS	7291	2007	256	256/2
Waveform	10000	5000	21	3.17

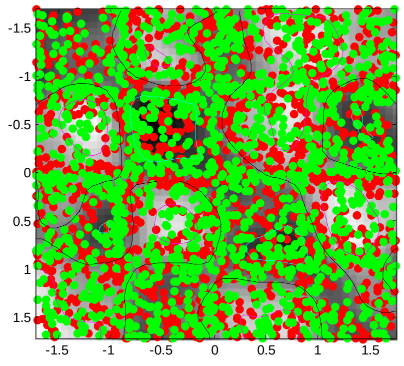
TABLE 2
ACCURACY($\times 100\%$) COMPARISON ON BENCHMARK DATA SETS

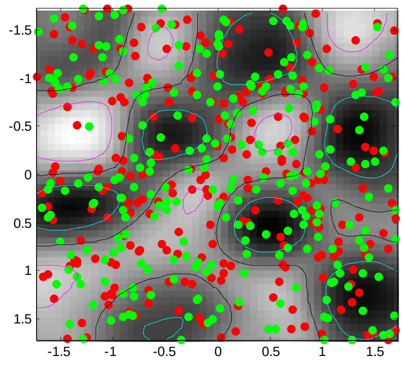
Data sets (#SVs)	*Perceptron* *B*= ∞	*Tighter*^{Full} *B=20(+N)*	Tighter^{B} *B=20(+20)*	Stoptron *B=20*	Forgetron *B=20*	Random *B=20*	Tighter^{0} *B=20*	Tightest *B=20*
Adult9 (6502)	78.0 ± 2.1	80.1 ± 2.1	75.5 ± 1.5	75.5 ± 2.9	67.8 ± 7.8	69.4 ± 12.8	69.5 ± 9.5	***80.8 ± 0.8***
Banana (582)	84.7 ± 1.9	87.6 ± 1.5	79.3 ± 2.6	79.2 ± 3.8	76.0 ± 4.3	74.5 ± 4.9	78.2 ± 3.3	***86.7 ± 1.9***
NCheckerb (3089)	79.8 ± 3.1	85.8 ± 1.2	62.6 ± 3.8	64.3 ± 4.8	60.2 ± 3.9	59.6 ± 4.3	62.7 ± 3.4	**77.7 ± 2.2**
Covertype (28056)	72.7 ± 4.5	61.3 ± 6.3	55.7 ± 4.7	55.9 ± 3.4	53.9 ± 3.8	53.3 ± 2.2	53.2 ± 2.1	**66.1 ± 1.1**
Gauss (2616)	72.6 ± 6.7	77.9 ± 4.1	67.3 ± 3.4	69.8 ± 4.8	66.6 ± 5.6	67.0 ± 3.4	64.6 ± 6.6	**80.8 ± 0.6**
IJCNN (2302)	96.2 ± 1.6	80.4 ± 12.0	80.3 ± 11.8	67.4 ± 17.2	77.7 ± 8.4	72.8 ± 23.3	82.0 ± 13.0	**87.8 ± 2.6**
Letter (1250)	95.9 ± 0.4	77.1 ± 1.6	63.7 ± 4.2	63.6 ± 2.2	61.6 ± 3.5	61.1 ± 2.8	60.0 ± 1.8	**67.1 ± 1.8**
MNIST (525)	97.4 ± 0.9	78.3 ± 15.6	76.8 ± 9.4	79.7 ± 10.8	72.1 ± 17.2	84.9 ± 5.2	82.0 ± 6.6	**87.8 ± 3.5**
Pendigits (248)	97.7 ± 1.3	83.6 ± 5.8	80.2 ± 6.9	80.7 ± 5.5	82.7 ± 6.7	80.4 ± 5.7	82.7 ± 5.1	**84.3 ± 4.8**
USPS (527)	94.5 ± 1.1	73.0 ± 7.1	74.5 ± 2.5	75.1 ± 3.2	65.4 ± 7.6	69.7 ± 6.4	70.3 ± 5.2	**78.7 ± 1.9**
Waveform (1482)	86.2 ± 0.7	86.8 ± 1.0	77.4 ± 2.4	77.5 ± 2.8	76.4 ± 4.8	75.6 ± 4.7	72.6 ± 3.6	**85.1 ± 1.5**
Average	87.0	78.5	71.6	71.1	68.4	69.3	70.5	**79.8**

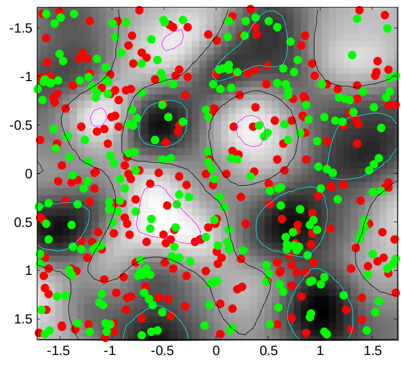
	B= ∞	*B=100(+N)*	*B=100(+100)*	*B=100*	*B=100*	*B=100*	*B=100*	*B=100*
Adult9 (6502)	78.0 ± 2.1	83.0 ± 0.9	76.4 ± 1.9	76.4 ± 3.0	75.1 ± 4.4	75.3 ± 4.0	71.2 ± 7.3	***81.7 ± 0.6***
Banana (582)	84.7 ± 1.9	89.0 ± 1.3	86.1 ± 2.3	85.2 ± 2.0	82.1 ± 5.9	82.1 ± 3.8	83.2 ± 3.5	***88.9 ± 0.8***
NCheckerb (3089)	79.8 ± 3.1	92.6 ± 0.6	74.2 ± 5.2	69.8 ± 3.8	66.4 ± 4.5	68.4 ± 3.0	71.4 ± 4.6	***87.6 ± 1.3***
Covertype (28056)	72.7 ± 4.5	64.3 ± 2.5	61.0 ± 3.2	61.5 ± 3.8	59.6 ± 2.9	59.2 ± 2.9	56.9 ± 3.5	**70.6 ± 1.1**
Gauss (2616)	72.6 ± 6.7	80.5 ± 0.6	74.3 ± 2.6	69.6 ± 7.3	72.2 ± 4.3	71.3 ± 5.0	65.7 ± 4.4	***80.8 ± 0.8***
IJCNN (2302)	96.2 ± 1.6	89.3 ± 8.5	90.3 ± 4.1	89.3 ± 3.5	87.9 ± 5.4	81.5 ± 9.9	82.9 ± 15.3	**91.7 ± 0.4**
Letter (1250)	95.9 ± 0.4	86.1 ± 0.5	74.6 ± 1.5	74.9 ± 1.8	72.5 ± 1.7	72.7 ± 1.7	73.3 ± 2.9	**79.6 ± 0.7**
MNIST (525)	97.4 ± 0.9	89.6 ± 5.0	91.5 ± 3.3	93.4 ± 4.3	92.8 ± 2.8	90.3 ± 4.7	80.0 ± 10.6	**95.8 ± 0.4**
Pendigits (248)	97.7 ± 1.3	93.7 ± 2.7	94.4 ± 2.1	96.2 ± 1.3	94.2 ± 4.0	98.3 ± 0.6	91.6 ± 4.2	**97.1 ± 0.8**
USPS (527)	94.5 ± 1.1	85.1 ± 4.4	85.2 ± 3.6	81.2 ± 8.9	81.0 ± 7.4	80.8 ± 7.1	76.8 ± 8.9	**89.2 ± 0.9**
Waveform (1482)	86.2 ± 0.7	88.0 ± 0.7	83.5 ± 1.6	83.9 ± 0.7	81.0 ± 1.2	82.1 ± 2.1	80.0 ± 1.8	***87.5 ± 0.3***
Average	87.0	85.3	80.8	79.8	78.4	78.0	75.3	**86.3**

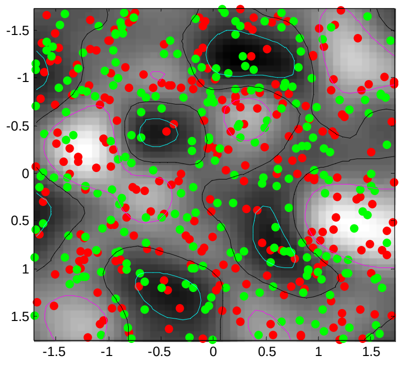
	B= ∞	*B=500(+N)*	*B=500(+500)*	*B=500*	*B=500*	*B=500*	*B=500*	*B=500*
Adult9 (6502)	78.0 ± 2.1	82.7 ± 0.3	79.0 ± 0.6	78.3 ± 1.2	76.3 ± 2.5	77.0 ± 3.6	69.4 ± 6.5	***82.4 ± 0.3***
Banana (582)	84.7 ± 1.9	88.9 ± 1.1	88.2 ± 1.3	87.5 ± 0.8	84.8 ± 2.3	85.3 ± 1.7	84.8 ± 2.5	***89.9 ± 1.0***
NCheckerb (3089)	79.8 ± 3.1	94.9 ± 1.0	87.6 ± 1.1	76.6 ± 2.8	70.3 ± 5.2	73.6 ± 4.3	75.5 ± 5.6	***94.2 ± 0.8***
Covertype (28056)	72.7 ± 4.5	71.9 ± 0.8	68.7 ± 3.5	65.3 ± 5.4	65.7 ± 1.9	62.9 ± 3.1	62.6 ± 3.4	***75.4 ± 0.6***
Gauss (2616)	72.6 ± 6.7	80.7 ± 0.4	78.1 ± 1.3	72.2 ± 3.1	70.1 ± 5.8	68.8 ± 5.0	65.6 ± 5.4	***81.4 ± 0.5***
IJCNN (2302)	96.2 ± 1.6	94.0 ± 2.8	94.5 ± 1.3	93.1 ± 4.4	90.1 ± 5.7	91.6 ± 2.4	93.5 ± 3.4	**94.3 ± 0.5**
Letter (1250)	95.9 ± 0.4	93.9 ± 0.6	89.6 ± 0.6	90.5 ± 0.3	88.0 ± 1.4	86.3 ± 1.0	88.2 ± 0.9	**91.6 ± 0.5**
MNIST (525)	97.4 ± 0.9	96.5 ± 1.3	95.6 ± 1.1	95.3 ± 5.9	95.2 ± 2.6	95.4 ± 2.3	95.2 ± 2.2	***97.5 ± 0.3***
Pendigits (248)	98.2 ± 0.6	98.2 ± 0.6	98.2 ± 0.6	98.2 ± 0.6	98.2 ± 0.5	98.2 ± 0.6	98.2 ± 0.6	***98.2 ± 0.6***
USPS (527)	94.5 ± 1.1	93.7 ± 2.7	93.2 ± 1.3	93.2 ± 1.7	90.5 ± 3.2	90.6 ± 4.3	92.5 ± 1.6	**94.7 ± 0.5**
Waveform (1482)	86.2 ± 0.7	87.9 ± 0.2	85.3 ± 0.8	85.0 ± 1.0	84.3 ± 1.2	83.8 ± 1.1	82.0 ± 1.0	***87.3 ± 0.6***
Average	87.0	89.5	87.3	85.0	82.9	83.0	82.6	***90.0***

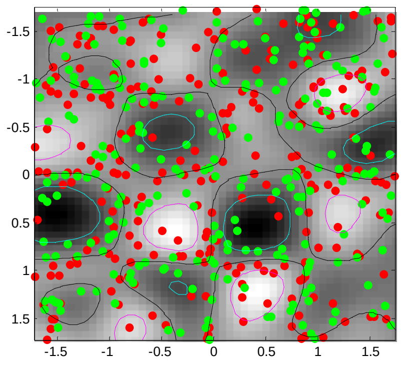
Values in parentheses in the data set column are # of SVs learned by Perceptron. Values in bold in Tightest column indicate the highest accuracy among budget Perceptron algorithms. Values in italics in Tightest column indicate the accuracy is even better than Perceptron. Values in parentheses in Tighter^{Full} and Tighter^{B} columns are the budget size for the additional validation set.

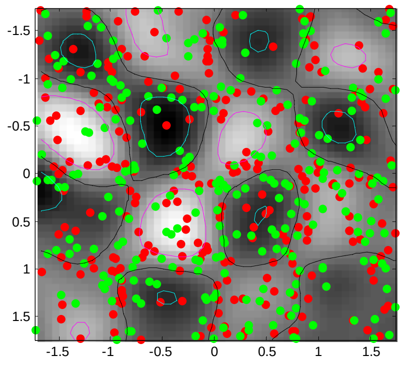

(a) *Perceptron* solution

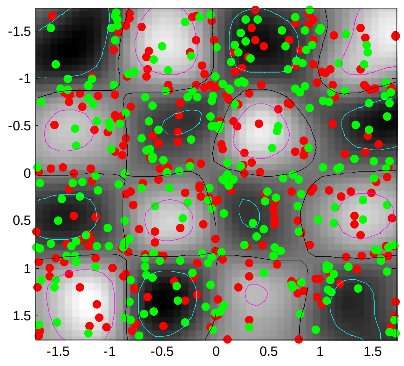

(b) Stoptron solution

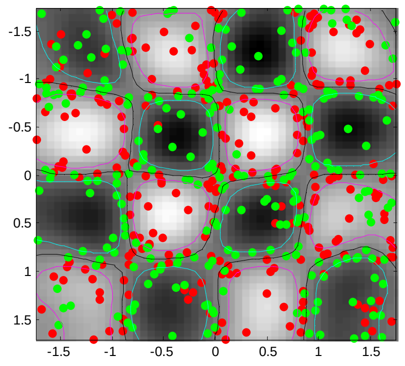

(c) Forgetron solution

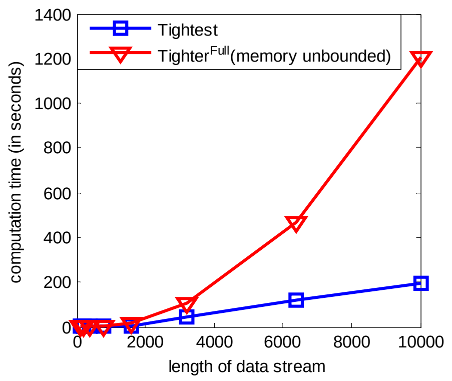

(d) Random solution


(e) Tighter^0 solution


(f) Tighter^B solution


(g) *Tighter^{Full}* solution


(h) Tightest solution



(i) Computation time comparison

Fig 2. Solutions of all algorithms on *NChecherboard* data

assignment was switched for 15% of the randomly selected examples. For both testing sets, we used the noise-free version as the test set. In this way, the highest reachable accuracy for *N-Checkerboard* was 100%.

B. Evaluation Procedure

We compared the proposed Tightest Perceptron algorithm with four state of the art budget perceptron algorithms: Self-Tuned *Forgetron* [6], *Random* Perceptron [2], and *Tighter*^{0} and *Tighter*^{A} Perceptrons [13], as well as to the baseline algorithm *Stoptron* where the kernel perceptron terminates once the budget is full. For Tighter^A, we use $A=B$ randomly selected examples as the additional validation set, and denote it as Tighter^B. As a reference, we also present results from the original Kernel *Perceptron*, and the budget unconstrained version of Tighter Perceptron, *Tighter^{Full}* [13] (names in *italics* are used in Table 2 and Figure 2).

We evaluated three different budgets B = 20, 100, 500, using an RBF kernel defined as $k(x,y) = exp(-||x-y||^2/2\delta^2)$, where δ is the RBF width. To keep things simple, for *Adult9*, *USPS* and *Waveform* we used the same kernel width as in previous papers [10, 13]. For 2-dimensional data sets, a small kernel width of 0.1 was used and for all the remaining data sets the kernel width was set to $\delta^2 = M/2$ [3], where M is the number of attributes. The summary of kernel widths is shown in Table 1. Training examples were ordered randomly.

Attributes in all data sets were scaled to mean zero and standard deviation one.

C. Results

In this section we summarize performance results on all 11 benchmark data sets. Each result (*mean* ± *std*) listed in Table 2, comparing the alternative kernel perceptron algorithms at three different budgets, is an average and standard deviation of 10 repeated experiments.

From Table 2 it can be seen that Tightest significantly outperforms all competing budget perceptron algorithms on every data set and for all three budgets. The Tightest is significantly more accurate than both Tighter^0 and Tighter^B that require roughly twice larger memory. This result confirms that using the posterior class probability by the proposed method provides highly valuable information for accuracy estimation.

It is worth noting that Tightest is often better than even the memory unbounded Tighter$^{\text{Full}}$. A part of the explanation for such behavior is that Tighter$^{\text{Full}}$ uses the 0-1 loss while Tightest uses the hinge loss that is more sensitive to the errors far from the decision boundary. Therefore, it may be more suitable for removing outlying noisy support vectors.

Comparing Tightest with the memory unbounded Kernel Perceptron, we can observe that Tightest is highly competitive and sometimes even more accurate than Kernel Perceptron. As seen, the accuracy of Tightest with B=500 is better than Perceptron in 8 of 11 data sets, with a modest budget B=100 Tightest is more accurate 5 times, and even with a tiny budget of B=20 Tightest still beats Perceptron on 3 of the noisiest data sets. The success of Tightest probably lies in its ability to remove less useful or even harmful support vectors after consulting the accuracy after removal.

Of the remaining results, it is interesting to note that the two theoretically well behaved algorithms Fogetron and Random had quite poor performance and it was comparable to Tighter^0. Their accuracy was often below the simple baseline algorithm Stoptron. This behavior is particularly noticeable on the noisiest data sets.

D. Illustration on 2D N-Checkerboard

In Figure 2 we illustrate the solutions of various algorithms on *NCheckerboard* data. Budget B=500 was used for the budget Perceptron algorithms. In Figure 2(a-h) magenta and cyan lines are positive and negative margins, respectively. Black line is the decision boundary, and red and green dots indicate positive and negative SVs, respectively. It can be seen that the decision boundaries created by Perceptron, Stoptron, Random, Forgetron and Tighter^0 in Figure 2(a-f) are not particularly successful, making it difficult to distinguish the underlying checkerboard. In contrast, Tighter$^{\text{Full}}$ and Tightest solutions are quite successful and it is easy to distinguish the checkerboard pattern. Another interesting observation is that the support vectors in the Tightest solution lie close to the decision boundary.

In Figure 2(i) the time comparison between the two optimal solution algorithms is illustrated. As seen, the memory bounded Tightest runtime appears linear while the memory unbounded Tighter$^{\text{Full}}$ runtime appears quadratic, as expected.

V. Conclusion

In this paper we presented the Tightest Perceptron algorithm for online learning on a budget. The algorithm achieves constant update runtime and constant space complexity with the training data size. Experimental results showed that Tightest significantly outperforms state-of-the-art budget perceptron algorithms and is often superior to the memory unbounded kernel perceptron, despite using a rather small budget. This hints at the possibility of building accurate perceptron classifiers from very large data streams while operating under a very limited memory budgets. Furthermore, Tightest results in very compact predictors and it directly addresses a problem often observed in practice where the size of the support vector set grows with the training data size.

References

[1] M. Aizerman, E. Braverman, and L. Rozonoer, "Theoretical foundations of the potential function method in pattern recognition learning," in Automation and Remote Control, 1964.

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

[3] C. Chang and C. Lin. LIBSVM: alibrary for support vector machines, 2001. Available: http://www.csie.ntu.edu.tw/ cjlin/libsvm/.

[4] K. Crammer and J. Kandola and Y. Singer, "Online classification on a budget," in *Advances in Neural Information Processing Systems*, 2004.

[5] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz and Y. Singer, "Online Passive-Aggressive Algorithms," in *Journal of Machine Learning Research*, 2006.

[6] O. Dekel and S. S. Shwartz and Y. Singer, "The Forgetron: A kernel-based Perceptron on a budget," in *SIAM Journal on Computing*, 2008.

[7] C. Gentile, "A New Approximate Maximal Margin Classification Algorithm," in *Journal of Machine Learning Research*, 2001.

[8] J. Kivinen, A. J. Smola, and R. C. Williamson, "Online Learning with Kernels," in *IEEE Transactions on Signal Processing*, 2001.

[9] Y. Li and P. Long , "The relaxed online maximum margin algorithm," in *Machine Learning*, 2002.

[10] F. Orabona, J. Keshet and B. Caputo, "The Projectron: a Bounded Kernel-Based Perceptron," in Internatinal Conference on Machine Learning, 2008.

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

[12] V. N. Vapnik, Statistical Learning Theory, John Wiley & Sons, Inc., 1995.

[13] J. Weston, A. Bordes and L. Bottou, "Online (and Offline) on an Even Tighter Budget," in *International Workshop on Artificial Intelligence and Statistics*, 2005.