COMPRESSED KERNEL PERCEPTRONS

Slobodan Vucetic* Vladimir Coric Zhuang Wang

Department of Computer and Information Sciences
Temple University
Philadelphia, PA 19122, USA
* email: vucetic@ist.temple.edu

Abstract

Kernel machines are a popular class of machine learning algorithms that achieve state of the art accuracies on many real-life classification problems. Kernel perceptrons are among the most popular online kernel machines that are known to achieve high-quality classification despite their simplicity. They are represented by a set of B prototype examples, called support vectors, and their associated weights. To obtain a classification, a new example is compared to the support vectors. Both space to store a prediction model and time to provide a single classification scale as $O(B)$. A problem with kernel perceptrons is that the number of support vectors tends to grow without bounds with the number of training examples on noisy data. To reduce the strain at computational resources, budget kernel perceptrons have been developed by upper bounding the number of support vectors. In this work, we proposed a new budget algorithm that upper bounds the number of bits needed to store kernel perceptron. Setting the bitlength constraint could facilitate development of hardware and software implementations of kernel perceptrons on resource-limited devices such as microcontrollers. The proposed compressed kernel perceptron algorithm decides on the optimal tradeoff between number of support vectors and their bit precision. The algorithm was evaluated on several benchmark data sets and the results indicate that it can train highly accurate classifiers even when the available memory budget drops below 1 Kbit. This promising result points to a possibility of implementing powerful learning algorithms even on the most resource-constrained computational devices.

1 Introduction

Kernel perceptrons are popular online binary classification algorithms that learn a mapping f: $X \rightarrow R$ from a stream of training examples $S = \{(x_t, y_t), t = 1 \ldots T\}$, where $x_t \in X$ is an M-dimensional input vector, called an instance, and $y_t \in \{-1, +1\}$ is a binary variable, called a label. Typically, the resulting kernel perceptron can be represented as

$$f(x) = \sum_{i=1}^{T} \alpha_i K(x_i, x), \tag{1}$$

where α_i are weights associated with training examples, and K is the kernel function. In practice, most weights become zero and the instances with nonzero weights are called *support vectors*. We note that the functional form (1) is common to several other machine learning algorithms such as support vector machines, radial basis functions, and Parzen

window classifiers. The classification of instance x by kernel perceptron is obtained as $sign(f(x))$ while $|f(x)|$ can be interpreted as the prediction confidence.

In its basic form, training of a kernel perceptron consists of a simple procedure: starting from the zero function $f(x) = 0$ at time $t = 0$, training examples are observed sequentially, and $f(x)$ is updated as $f(x) \leftarrow f(x) + \alpha_t K(x_t, x)$ where $\alpha_t = y_t$ every time the instance x_t is misclassified, i.e. when $y_t f(x_t) \leq 0$. Despite the simplicity of this training algorithm, kernel perceptrons often achieve impressive classification accuracies on highly nonlinear problems. On the other hand, training and use of kernel perceptrons can place a surprisingly heavy burden on computational resources. The main reason for this is that the number of support vectors grows proportionally with the number of training examples in noisy classification problems [3]. Therefore, the space required to store kernel perceptron is $O(TM)$, the time required to make a single prediction is also $O(TM)$, and the training time is $O(T^2M)$.

The budget kernel perceptrons [5] have been developed to address the problem of the unbounded growth in resource consumption with training data size. Their basic idea is to maintain a constant number of support vectors during the training by removing a single support vector every time the budget is exceeded upon addition of a new support vector. Given a budget of B support vectors, they achieve constant space scaling $O(BM)$ and linear training time $O(TBM)$. Among the proposed removal approaches are removal of a randomly selected support vector (called Random Perceptron) [4], the oldest support vector [6], or the support vector with the largest $\alpha_t f(x_t)$ [9]. The experimental results indicate that reasonably accurate kernel perceptrons can be trained using quite modest budgets. More recently, a related algorithm was proposed [7] that has an option to project the new support vector to the existing ones instead of adding it, whenever it can be done with a sufficiently small degradation of the classifier. While it shows an improved classification accuracy for the same budget, its space scaling is $O(B^2M)$ and training time is $O(TB^2M)$. In this sense, this algorithm could support only the budget of $\sqrt{B}$ support vectors to maintain the same performance as the algorithms with budget of B that use removal.

In this paper, we propose a kernel perceptron algorithm that defines budget in terms of the available bitlength L instead of the number of support vectors. The budget expressed in bitlength can be directly linked to the hardware-level memory capacity and can be appropriate when implementing kernel perceptrons on low-resource devices such as microcontrollers. Constraining the memory budget in this way opens an interesting question of a tradeoff between the number of support vectors and their arithmetic precision. More specifically, the bitlength needed to store a kernel perceptron can be expressed as the sum

$$L = B \cdot M \cdot b_x + B \cdot b_\alpha + l_a \qquad (2)$$

where b_x is the precision for each of the M instance attributes, b_α is the precision for support vector weights, and l_a is the bitlength needed for the ancillary variables.

In this study we focus on Random Perceptron due to its simplicity and a very frugal use of computational resources (e.g. only one bit is needed to store each support vector weight). We will show that the best choice of B and b_x depends on the budget L and the

choice of kernel function K. Using the estimations for quality loss due to support vector removal and loss due to loss of precision, the proposed Compression Kernel Perceptron is a modification of the Random Perceptron that allows determination of the optimal values of B and b_x during training. We will experimentally illustrate on several benchmark data sets that the proposed algorithm can train accurate classifiers even when the available memory budget is severely limited.

It should be noted that a significant body of research exists on the related topic of efficient hardware implementations of various machine learning and signal processing algorithms. In the area of kernel machines, for example, solutions have been proposed for implementations on fixed point processors of support vector machines [1] and kernel perceptrons [2]. The main focus of these and similar studies was on modifying the original algorithms to reducing the quantization effects of calculations. However, these studies do not address the memory constraint problem and the proposed algorithms could be used only when training data sets are small. In our work we assume availability of a floating point processor and focus of optimizing quality of learning given the limited memory. Since our algorithm typically results in kernel perceptrons represented with a precision of only a few bits, it is evident that it could be possible to implement it on a fixed-point processor without much loss in the performance.

2 Preliminaries

The classical perceptron [8] is a linear function of type $f(x) = w^{\mathrm{T}} x$ that starts with $f(x) = 0$, and updates it as $f(x) \leftarrow f(x) + y_t x_t$ every time the instance x_t is misclassified, $y_t f(x_t) \leq 0$. As an alternative to the classical perceptron, instances x can be first mapped to feature vectors $\Phi(x)$ and the classical perceptron can be trained on the feature vectors. The resulting perceptron will be a linear function in the feature space,

$$f(x) = {w_\Phi}^T \Phi(x), \tag{3}$$

where w_Φ is a vector in the feature space expressed as

$$w_\Phi = \sum_{i=1}^{T} \alpha_i \Phi(x_i). \tag{4}$$

If mapping Φ is properly chosen, $f(x)$ could solve nonlinear classification problems. Kernel perceptrons are a special type of perceptrons that operate in the feature space, where Φ is induced by a Mercer kernel K such that $\Phi(x)^{\mathrm{T}}\Phi(z) = K(x, z)$. The main benefit of using this special class of mappings is that the resulting perceptron can be expressed as (1), so that there is no need to explicitly work in the, potentially highly dimensional, feature space. Among many possible kernel functions, the most popular are linear kernel, $K(x, z) = x^{\mathrm{T}} z$, that results in the classical perceptron, and Radial Basis Function (RBF) kernel, $K(x, z) = exp\,(-\|x - z\|^2 / A^2)$, where A is the kernel width, that is known to achieve state of the art accuracy on many highly nonlinear problems. It is interesting to note that the RBF kernel induces an infinitely dimensional feature space.

The Compressed Kernel Perceptron proposed in this paper is based on Random Perceptron [4], a budget kernel perceptron with random removal of support vectors when budget is exceeded. This choice was based on the simplicity of the algorithm, which is

fundamental in severely resource constrained applications. Moreover, Random Perceptron achieves similar performance and has the same error bound as the more involved budget perceptrons [4]. From the memory budget viewpoint, Random Perceptron is very desirable because support vector weights are binary (they equal the class labels) and so can be stored with a single bit. This represents a significant advantage over Forgetron [6] that removes the oldest support vector. Forgetron subjects weights to an exponential decay that requires multi-bit representation of weights. As a result, Forgetron can have fewer support vectors given the same memory budget. Random Perceptron is also more efficient than Tighter Perceptron [9], a popular budget perceptron algorithm that removes support vector with the highest value of $\alpha_t f(x_t)$ (being the most confidently correctly classified support vector). In this case, the weights remain binary as in Random perceptrons, but there is a significant computational overhead in searching for the best support vector to remove. It turns out that this requires either constant space and $O(BK)$ time for every removal, or $O(B)$ time and space, which are both inferior to constant time and space required for removal of a random support vector.

3 Methodology

The main dilemma in the Compressed Kernel Perceptron is what to do when the memory budget is exceeded and when a new support vector should be added. One option is to reduce precision of the existing support vectors to accommodate for the new support vector and another is to replace a randomly selected support vector with the new one (as is done in Random perceptron). To answer this question, we will derive expressions for the loss due to precision reduction, $loss_Q$ (Section 3.1), and the loss due to replacement of a support vector, $loss_R$ (Section 3.2). Given the values of $loss_Q$ and $loss_R$, the Compressed Kernel Perceptron can be described with a pseudo code in Table 1.

We define the loss as the squared norm of the difference between weights of the desired and the damaged kernel perceptron,

$$loss = \| w_\Phi - w_\Phi' \|^2, \tag{5}$$

where w_Φ is the (potentially infinitely dimensional) perceptron weight and w_Φ' is the same weight after the decrease in precision of support vectors or after the removal of a support vector. In the following, we will focus on loss for kernel perceptrons with RBF kernels.

3.1. Quantization loss

Let us first calculate loss of kernel perceptron when attributes $[x_{t1}\ x_{t2}\ \dots\ x_{tM}]$ of instance x_t are quantized as $Q(x_t) = [Q(x_{t1})\ Q(x_{t2})\ \dots\ Q(x_{tM})]$ to b-bit precision. We will assume,that all attributes are scaled to range between [0, 1] and that quantization is uniform (each of the 2^b codewords represents a bin of the same length 2^{-b}). In this case, the quantization error of m-th attribute, $\Delta_{tm} = x_{tm} - Q(x_{tm})$, is uniformly distributed in the range between $-2^{-(b+1)}$ and $2^{-(b+1)}$. The quantization loss for reducing precision from infinite to b-bit can be expressed

$$Loss_q(b) = E\left\{ \| \sum_{t=1}^{T} \alpha_t \Phi(x_t) - \sum_{t=1}^{T} \alpha_t \Phi(Q(x_t)) \|^2 \right\} = \sum_{t=1}^{T} {\alpha_t}^2 \cdot E\left\{ \| \Phi(x_t) - \Phi(Q(x_t)) \|^2 \right\}$$
$$= \sum_{t=1}^{T} {\alpha_t}^2 \cdot E\{K(x_t, x_t) + K(Q(x_t), Q(x_t)) - 2K(x_t, Q(x_t))\}. \quad (6)$$

When the feature space Φ is induced by RBF kernel with width A, (6) can be written as

$$Loss_q(b) = \sum_{t=1}^{T} {\alpha_t}^2 \cdot 2E\{1 - K(x_t, Q(x_t))\} = \sum_{t=1}^{T} {\alpha_t}^2 \cdot 2E\left\{ 1 - \exp(-\frac{\| x_t - Q(x_t) \|^2}{A^2}) \right\}. \quad (7)$$

The difference $x_t - Q(x_t)$ is an M-dimensional random vector whose elements are independent random variables with uniform distribution $U(-2^{-(b+1)}, 2^{-(b+1)})$. Therefore, we can write

$$E\left\{ 1 - \exp(-\frac{\| x_t - Q(x_t) \|^2}{A^2}) \right\} = 1 - \left(E\left\{ \exp(-\varepsilon^2) \right\}^M \right), \quad (8)$$

where $\varepsilon \sim U(-a, a)$, and a is defined as $a = 2^{-(b+1)}/A$.

Since $E\left\{ \exp(-\varepsilon^2) \right\} = erf(a) \cdot \sqrt{\pi} / 2a$, the quantization loss can be expressed as

$$Loss_q(b) = 2 \cdot \left(1 - \left(erf(a) \cdot \sqrt{\pi} / 2a \right)^M \right) \cdot \sum_{t=1}^{T} {\alpha_t}^2. \quad (9)$$

The quantization loss $Loss_q(b)$ can be directly used in deriving $loss_q$ incurred by loss of precision needed for inclusion of a new support vector. Let us, for a moment, neglect quantity l_a from (2) and observe that in Random Perceptrons $\alpha_t \in \{-1, +1\}$ for B support vectors and $\alpha_t = 0$ otherwise (thus, the sum $\Sigma {\alpha_t}^2$ in (9) equals B), and that $b_\alpha = 1$. From (2) it follows that the allowed bit precision for kernel perceptron with B support vectors equals $b = (L/B - 1)/M$. The value of $loss_q(B)$ due to an increase from B to $(B+1)$ support vectors can be expressed as

$$loss_q(B) = Loss_q\left((L/(B+1) - 1)/M \right) - Loss_q\left((L/B - 1)/M \right) \quad (10)$$

We observe that equation (10) allows for fractional number of bits. This is not an issue, because $loss_q$ gives the desired information about expected distortion of kernel perceptron due to precision decrease upon addition of a single support vector. In our implementation, we address the issue of fractional number of bits b by representing fraction $\lceil b \rceil - b$ of instances with b-bit precision and fraction $b - \lfloor b \rfloor$ of instances with $(b+1)$-bit precision. Upon addition of the $(B+1)$-st support vector, precision of additional b instances is reduced from $b+1$ to b.

Table 1. Compressed Kernel Perceptron

Input: a data sequence (x_1, y_1), ... (x_T, y_T), kernel K, bitlength L

Initialize: $f(x) = y_1 K(x_1, x)$, $B = 1$, $b = L/M$

Output: $f(x) = \sum_{i=1}^{T} \alpha_i K(x_i, x)$,

for t = 2, 3, … T

  if $y_t f(x_t) \le 0$

    calculate $loss_Q$ (eq. 10) and $loss_R$ (eq. 12)

    if $loss_Q < loss_R$

      $b \leftarrow L / M(B+1)$

      reduce precision of support vectors to b bits

      $f(x) \leftarrow f(x) + \alpha_t K(x_t, x)$

    else

      $j \leftarrow$ index of a randomly selected support vector

      $f(x) \leftarrow f(x) + \alpha_t K(x_t, x) - \alpha_j K(x_j, x)$,

3.2 Removal loss

Let us now consider the loss after a randomly selected support vector is removed from the kernel perceptron. The resulting perceptron w_Φ' after removal of support vector x_i from kernel perceptron w_Φ can be expressed as $w_\Phi' = w_\Phi - \alpha_i \Phi(x_i)$. Naively, we can attempt to express the removal loss as $loss_R = \|w_\Phi - w_\Phi'\|^2 = \|\alpha_i \Phi(x_i)\|^2$, which results in constant loss $loss_R = 1$ for Random Perceptrons with RBF kernels. However, such estimation of removal loss is inappropriate.

To illustrate this, let us consider cumulative loss due to removal of two support vectors, x_i and x_j. The removal loss in this case is $loss_R = \|\alpha_i \Phi(x_i) + \alpha_j \Phi(x_j)\|^2 = 2 + 2\alpha_i \alpha_j K(x_i, x_j)$. Therefore, if kernel width is large (which is quite a common choice in kernel perceptrons) $K(x_i, x_j)$ could be close to 1 and $loss_R$ would be either close to 0 (if α's have opposite sign) or 4 (if α's have equal sign). Thus, the $loss_R$ of removal of a single support vector would be near 0 or 2, which is very different from the previous conclusion that it is always 1.

Most real life classification data sets are noisy. Let us assume the minimal achievable (Bayes) error on a given data set is e. Then, the probability that a newly observed data point will become support vector is at least e. Given the random removal process of Random Perceptron, it can practically be guaranteed that every support vector currently in the kernel perceptron will eventually be removed. Therefore, the best possible estimate of the removal loss can be obtained by considering average loss after all support vectors currently in the perceptron are removed. Therefore, the removal loss can be estimated as

the average norm of the perceptron,

$$loss_R = \frac{1}{B}\left\| \sum_{i=1}^{B} \alpha_i \Phi(x_i) \right\|^2 = \frac{1}{B}\sum_{i=1}^{B}\sum_{j=1}^{B} \alpha_i \alpha_j K(x_i, x_j) = \frac{1}{B}\sum_{i=1}^{B} y_i f(x_i), \qquad (11)$$

where $f(x)$ is the prediction of a current kernel perceptron. We observe that the removal loss is equal to the average of prediction margins of the kernel perceptron on its support vectors.

We propose to evaluate $loss_R$ using (11) after every update of kernel perceptron. A potential issue with this strategy is computational effort needed to evaluate (11) that seems to require $O(B^2M)$ time. However, with some care, the time overhead for calculation of $loss_R$ can be reduced to retain the computational cost of the Compressed Kernel Perceptron on par with the Random Perceptron. By assuming that $loss_R^{old}$ is estimate from the previous perceptron update, the $loss_R^{new}$ of the new perceptron where new support vector (x_{new}, y_{new}) replaces the old one (x_R, y_R) can be calculated as

$$loss_R^{new} = loss_R^{old} + 2/B \cdot (y_{new} f(x_{new}) - y_R f(x_R) - y_{new} y_R K(x_{new}, x_R) + 1). \qquad (12)$$

Since $y_{new} f(x_{new})$ is already known (was needed to evaluate if x_{new} should become a support vector), the main additional effort is to calculate $y_R f(x_R)$ which takes the modest $O(BM)$ time.

4 Experiments

Experimental setup. We evaluated Compressed Kernel Perceptron on several benchmark classification datasets from UCI ML Repository whose properties are summarized in Table 2. A few of the data sets that were originally multi-class were converted to two-class data sets as follows. For the digit dataset *Pendigits* we converted classes representing digits 1, 2, 4, 5, 7 (non-round digits) to the negative class and those representing digits 3, 6, 8, 9, 0 (round digits) to the positive class. *Shuttle* data set was converted to binary data by representing class 1 as positive and the remaining ones as negative. Class 1 in the 3-class *Waveform* data set was treated as negative and the remaining two as positive. *Banana* was originally a binary class data set. Attributes in all data sets were scaled to the range between [0, 1].

In the experiments, we compared the standard kernel perceptron and the proposed Compressed Kernel Perceptron, both using RBF kernels of width A (listed in Table 2). Because the kernel perceptron is a memory-unconstrainted online algorithm it serves as an upper bound on achievable accuracy. Additionally, it provides useful information about the desired number of support vectors when computational resources are not an issue. For experiments with Compressed Kernel Perceptrons, we evaluated five different budgets, $L = M \times \{50, 100, 200, 500, 1000\}$. Here, we counted only bitlength memory needed to store support vectors. Following (2) we should also add on top of this B bits for support vector weights and about 200 additional bits for the ancillary variables (this is the number in our current implementation). All experiments were repeated 10 times and the average and standard deviations are reported.

Table 2. Data set summaries

Data sets	**Training set**	**Test set**	*M*	*A*
Banana	4,300	1,000	2	0.1
Shuttle	42,603	14,167	9	0.1
Pendigits	7,494	3,498	16	1
Waveform	10,000	5,000	21	1

Table 3. Results on benchmark data sets. Average results over 10 repetitions are given. Numbers in parentheses are standard deviations

	Compressed Kernel Perceptron					**Perceptron**
	L/*M*					*L*
	50	100	200	500	1000	∞
Banana						
Accuracy	72.5 (6.9)	75.2 (10.0)	75.3 (5.0)	83.6 (4.6)	84.0 (2.5)	86.5 (1.5)
B	14.8 (0.4)	27.6 (0.5)	52.6 (1.9)	120.2 (0.8)	229 (4.5)	600.0 (20.0)
b	3.4 (0.1)	3.6 (0.1)	3.8 (0.1)	4.2 (0.0)	4.4 (0.1)	64
Shuttle						
Accuracy	93.2 (8.0)	95.0 (35.0)	96.7 (1.3)	97.4 (0.8)	98.1 (1.8)	99.6 (0.5)
B	11 (0.0)	21 (0.0)	40.2 (1.3)	91.6 (0.9)	175.6 (1.9)	262.0 (44.2)
b	4.5 (0.0)	4.7 (0.0)	5.0 (0.1)	5.5 (0.1)	5.7 (0.1)	64
Pendigits						
Accuracy	82.6 (5.2)	86.6 (4.9)	90.6 (2.4)	93.6 (3.5)	98.1 (1.3)	98.3 (0.3)
B	40.0 (0.0)	74.4 (1.3)	142.8 (7.2)	198.8 (0.4)	210 (0.0)	204.6 (4.7)
b	1.3 (0.0)	1.3 (0.0)	1.4 (0.0)	2.0 (0.0)	4.8 (0.0)	64
Waveform						
Accuracy	75.1 (6.3)	75.1 (5.4)	79.6 (3.0)	82.0 (4.2)	84.0 (6.9)	85.1 (3.6)
B	35.6 (0.6)	62.8 (0.4)	106.2 (8.0)	217.0 (3.8)	408.0 (7.7)	1473.2 (69.0)
b	1.4 (0.0)	1.6 (0.0)	1.9 (0.1)	2.3 (0.0)	2.4 (0.0)	64

Results. In Table 3 we provide information about accuracy, number of support vectors, and their bit precision for Compressed Kernel Perceptrons trained on benchmark data sets. Accuracy results for the standard kernel perceptron are shown in the last column of Table 3. It is evident that accuracies of Compressed Kernel Perceptron are very competitive to its memory unbounded counterpart. As expected, the most constrained scenario with 50 bits per attribute (i.e., 100 bits total for *Banana*, 450 for *Shuttle*, 800 for *Pendigits*, and 1050 for *Waveform* data set) resulted in considerably less accurate classification than the kernel perceptron. This is understandable since kernel perceptron needed hundreds of support vectors. However, the difference was not too large and was in all cases much higher than the 50% of the trivial predictor. Clearly, as the memory budget increased, the accuracy of our algorithm slowly approached that of kernel perceptron.

Considering the precision of support vectors, we observe that it gradually increased with the memory budget. This behavior is an expected consequence of the tradeoffs between number of support vectors and their precision estimated by equations (10) and (12).

5 Conclusions

In this study, we proposed the Compressed Kernel Perceptron algorithm that allows training of kernel perceptrons on devices with extremely limited memory budgets. The algorithm is based on Random Perceptron, a simple and memory friendly online learning algorithm that keeps number of support vectors constant by removing a random support vector upon addition of a new support vector. Compressed kernel perceptron estimates its distortions due to removal of a support vector and reduction in bit precision of support vectors and decides on the optimal tradeoff between number of support vectors and their precision. The experimental results showed that accurate classifiers could be learned efficiently while consuming very little memory.

The results are also useful in establishing lower bounds on memory needed to build an accurate classifier. They indicate that this lower bound is clearly a function of problem complexity and dimensionality. On the data dimensionality side, it is possible that careful attribute selection or transformation could lead to decrease in number of attributes and improve utilization of the available memory. What remains an open problem is whether it could be possible to perform attribute selection as part of the memory-constrained online algorithm without introducing a significant time and memory overhead.

In equation (2) we mentioned ancillary variables but we did not discuss them further. In our implementation of Compressed Kernel Perceptron, we require several ancillary variables that are in single precision format. These include variables to calculate kernel distance, quantization and removal loss, to determine perceptron prediction, and perform precision reduction. By observing that some of these variables can be reused for different purposes, our current implementation requires 6 double and 2 integer ancillary variables which introduces an overhead of about 200 additional bits. In extremely memory-limited applications it would be interesting to explore if this overhead could be reduced further.

There are many avenues for future research. One is implementation of Compressed Kernel Perceptron on floating and fixed point microcontrollers. Another is exploring how

to incorporate some critical machine learning operations such as attribute selection and kernel size selection in the algorithm, thus producing a truly adaptive learning agent. Finally, it would be interesting to explore if similarly successful memory-efficient algorithms could be developed for other types of kernel-based machine learning algorithms in problems of binary and multi-class classification, regression, clustering, and density estimation.

Acknowledgement

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

References

[1] D. Anguita, A. Boni, and S. Ridella. A digital architecture for support vector machines: theory, algorithm, and FPGA implementation. *IEEE Transactions on Neural Networks* 14: 5, 993-1009, 2003.

[2] D. Anguita, A. Boni, and S. Ridella. Digital kernel perceptron. *Electronics Letters*, 38: 10, 445-446, 2002.

[3] C. Burges. Simplified support vector decision rules. In *Proceedings of International Conference on Machine Learning*, 1996.

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

[5] K. Crammer, J. Kandola, and Y. Singer. Online classification on a budget. In *Advances in Neural Information Processing Systems* 16, 2004.

[6] O. Dekel, S. Shalev-Shwartz, and Y. Singer. The Forgetron: A kernel-based perceptron on a fixed budget. In *Advances in Neural Information Processing Systems* 18, 2005.

[7] F. Orabona, J. Keshet, B. Caputo. The Projectron: a bounded kernel-based perceptron. In *Proceedings of International Conference on Machine Learning,* 2008.

[8] F. Rosenblatt. The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* 65, 386–407, 1958.

[9] J. Weston, A. Bordes, and L. Bottou. Online (and offline) on an even tighter budget. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, 2005.