# Sublinear Optimization for Machine Learning

Kenneth L. Clarkson
*IBM Almaden Research Center*
*San Jose, CA*

Elad Hazan*
*Department of Industrial Engineering*
*Technion - Israel Institute of Technology*
*Haifa 32000 Israel*

David P. Woodruff
*IBM Almaden Research Center*
*San Jose, CA*

***Abstract*—We give sublinear-time approximation algorithms for some optimization problems arising in machine learning, such as training linear classifiers and finding minimum enclosing balls. Our algorithms can be extended to some kernelized versions of these problems, such as SVDD, hard margin SVM, and $L_2$-SVM, for which sublinear-time algorithms were not known before. These new algorithms use a combination of a novel sampling techniques and a new multiplicative update algorithm. We give lower bounds which show the running times of many of our algorithms to be nearly best possible in the unit-cost RAM model. We also give implementations of our algorithms in the semi-streaming setting, obtaining the first low pass polylogarithmic space *and* sublinear time algorithms achieving arbitrary approximation factor.**

## I. Introduction

Linear classification is a fundamental problem of machine learning, in which positive and negative examples of a concept are represented in Euclidean space by their feature vectors, and we seek to find a hyperplane separating the two classes of vectors.

The Perceptron Algorithm for linear classification is one of the oldest algorithms studied in machine learning [1], [2]. It can be used to efficiently give a good approximate solution, if one exists, and has nice noise-stability properties which allow it to be used as a subroutine in many applications such as learning with noise [3], [4], boosting [5] and more general optimization [6]. In addition, it is extremely simple to implement: the algorithm starts with an arbitrary hyperplane, and iteratively finds a vector on which it errs, and moves in the direction of this vector by adding a multiple of it to the normal vector to the current hyperplane.

The standard implementation of the Perceptron Algorithm must iteratively find a "bad vector" which is classified incorrectly, that is, for which the inner product with the current normal vector has an incorrect sign. Our new algorithm is similar to the Perceptron Algorithm, in that it maintains a hyperplane and modifies it iteratively, according to the examples seen. However, instead of explicitly finding a bad vector, we run another *dual* learning algorithm to learn the "most adversarial" distribution over the vectors, and use that distribution to generate an "expected bad" vector. Moreover, we do not compute the inner products with the current normal vector exactly, but instead estimate them using a fast sampling-based scheme.

Thus our update to the hyperplane uses a vector whose "badness" is determined quickly, but very crudely. We show that despite this, an approximate solution is still obtained in about the same number of iterations as the standard perceptron. So our algorithm is faster; notably, it can be executed in time *sublinear* in the size of the input data, and still have good output, with high probability. (Here we must make some reasonable assumptions about the way in which the data is stored, as discussed below.)

This technique applies more generally than to the perceptron: we also obtain sublinear time approximation algorithms for the related problems of finding an approximate Minimum Enclosing Ball (MEB) of a set of points, and training a Support Vector Machine (SVM), in the hard margin or $L_2$-SVM formulations.

We give lower bounds that imply that our algorithms for classification are best possible, up to polylogarithmic factors, in the unit-cost RAM model, while our bounds for MEB are best possible up to an $\tilde{O}(\varepsilon^{-1})$ factor. For most of these bounds, we give a family of inputs such that a single coordinate, randomly "planted" over a large collection of input vector coordinates, determines the output to such a degree that all coordinates in the collection must be examined for even a $2/3$ probability of success.

We show that our algorithms can be implemented in the parallel setting, and in the semi-streaming setting; for the latter, we need a careful analysis of arithmetic precision requirements and an implementation of our primal-dual algorithms using lazy updates, as well as some recent sampling technology [7].

Our approach can be extended to give algorithms for the kernelized versions of these problems, for some popular kernels including the Gaussian and polynomial, and also easily gives Las Vegas results, where the output guarantees always hold, and only the running time is probabilistic. [1] Our approach also applies to the case

*Work done while at IBM Almaden Research Center

[1]For MEB and the kernelized versions, we assume that the Euclidean norms of the relevant input vectors are known. Even with the addition of this linear-time step, all our algorithms improve on prior bounds, with the exception of MEB when $M = o(\varepsilon^{-3/2}(n+d))$.

of soft margin SVM (joint work in progress with Nati Srebro).

Our main results, except for semi-streaming and parallel algorithms, are given in Figure 1. The notation is as follows. All the problems we consider have an $n \times d$ matrix $A$ as input, with $M$ nonzero entries, and with each row of $A$ with Euclidean length no more than one. The parameter $\epsilon > 0$ is the additive error; for MEB, this can be a relative error, after a simple $O(M)$ preprocessing step. We use the asymptotic notation $\tilde{O}(f) = O(f \cdot \text{polylog}\frac{nd}{\varepsilon})$. The parameter $\sigma$ is the *margin* of the problem instance, explained below. The parameters $s$ and $q$ determine the standard deviation of a Gaussian kernel, and degree of a polynomial kernel, respectively.

The time bounds given for our algorithms, except the Las Vegas ones, are under the assumption of constant error probability; for output guarantees that hold with probability $1 - \delta$, our bounds should be multiplied by $\log(n/\delta)$.

The time bounds also require the assumption that the input data is stored in such a way that a given entry $A_{i,j}$ can be recovered in constant time. This can be done by, for example, keeping each row $A_i$ of $A$ as a hash table. (Simply keeping the entries of the row in sorted order by column number is also sufficient, incurring an $O(\log d)$ overhead in running time for binary search.)

By appropriately modifying our algorithms, we obtain algorithms with very low pass, space, and time complexity. Many problems cannot be well-approximated in one pass, so a model permitting a small number of passes over the data, called the semi-streaming model, has gained recent attention [11], [12]. In this model the data is explicitly stored, and the few passes over it result in low I/O overhead. It is quite suitable for problems such as MEB, for which any algorithm using a single pass and sublinear (in $n$) space cannot approximate the optimum value to within better than a fixed constant [13]. Unlike traditional semi-streaming algorithms, we also want our algorithms to be sublinear time, so that in each pass only a small portion of the input is read.

We assume we see the points (input rows) one at a time in an arbitrary order. The space is measured in bits. For MEB, we obtain an algorithm with $\tilde{O}(\varepsilon^{-1})$ passes, $\tilde{O}(\varepsilon^{-2})$ space, and $\tilde{O}(\varepsilon^{-3}(n+d))$ total time. For linear classification, we obtain an algorithm with $\tilde{O}(\varepsilon^{-2})$ passes, $\tilde{O}(\varepsilon^{-2})$ space, and $\tilde{O}(\varepsilon^{-4}(n+d))$ total time. For comparison, prior streaming algorithms for these problems [13], [14] require a prohibitive $\Omega(d)$ space, and none achieved a sublinear $o(nd)$ amount of time. Further, their guarantee is an approximation up to a fixed constant, rather than for a general $\varepsilon$ (though they can achieve a single pass).

*Formal Description: Classification:* In the linear classification problem, the learner is given a set of $n$ labeled examples in the form of $d$-dimensional vectors, comprising the input matrix $A$. The labels comprise a vector $y \in \{+1, -1\}^n$.

The goal is to find a separating hyperplane, that is, a normal vector $x$ in the unit Euclidean ball $\mathbb{B}$ such that for all $i$, $y(i) \cdot A_i x \geq 0$; here $y(i)$ denotes the $i$'th coordinate of $y$. As mentioned, we will assume throughout that $A_i \in \mathbb{B}$ for all $i \in [n]$, where generally $[m]$ denotes the set of integers $\{1, 2, \ldots, m\}$.

As is standard, we may assume that the labels $y(i)$ are all 1, by taking $A_i \leftarrow -A_i$ for any $i$ with $y(i) = -1$. The approximation version of linear classification (which is necessary in case there is noise), is to find a vector $x_\varepsilon \in \mathbb{B}$ that is an *$\varepsilon$-approximate solution*, that is,

$$\forall i' \;\; A_{i'} x_\varepsilon \geq \max_{x \in \mathbb{B}} \min_i A_i x - \varepsilon. \tag{1}$$

The optimum for this formulation is obtained when $\|x\| = 1$, except when no separating hyperplane exists, and then the optimum $x$ is the zero vector.

Note that $\min_i A_i x = \min_{p \in \Delta} p^\top A x$, where $\Delta \subset \mathbb{R}^n$ is the unit simplex $\{p \in \mathbb{R}^n \mid p_i \geq 0, \sum_i p_i = 1\}$. Thus we can regard the optimum as the outcome of a game to determine $p^\top A x$, between a minimizer choosing $p \in \Delta$, and a maximizer choosing $x \in \mathbb{B}$, yielding

$$\sigma \equiv \max_{x \in \mathbb{B}} \min_{p \in \Delta} p^\top A x,$$

where this optimum $\sigma$ is called the *margin*. From standard duality results, $\sigma$ is also the optimum of the dual problem

$$\min_{p \in \Delta} \max_{x \in \mathbb{B}} p^\top A x,$$

and the optimum vectors $p^*$ and $x^*$ are the same for both problems.

The classical Perceptron Algorithm returns an $\varepsilon$-approximate solution to this problem in $\frac{1}{\varepsilon^2}$ iterations, and total time $O(\varepsilon^{-2}M)$.

For given $\delta \in (0, 1)$, our new algorithm takes $O(\varepsilon^{-2}(n+d)(\log n)\log(n/\delta))$ time to return an $\varepsilon$-approximate solution with probability at least $1 - \delta$. Further, we show this is optimal in the unit-cost RAM model, up to poly-logarithmic factors.

*Formal Description: Minimum Enclosing Ball (MEB):* The MEB problem is to find the smallest Euclidean ball in $\mathbb{R}^d$ containing the rows of $A$. It is a special case of quadratic programming (QP) in the unit simplex, namely, to find $\min_{p \in \Delta} p^\top b + p^\top A A^\top p$, where $b$ is an $n$-vector. This relationship, and the generalization of our MEB algorithm to QP in the simplex, is discussed in §III-C; for more general background on QP in the simplex, and related problems, see for example [10].

### A. Related work

Perhaps the most closely related work is that of Grigoriadis and Khachiyan [15], who showed how to approximately solve a zero-sum game up to additive precision $\varepsilon$ in time $\tilde{O}(\varepsilon^{-2}(n+d))$, where the game matrix is $n \times d$. This problem is analogous to ours, and our algorithm is similar in structure to theirs, but where

| Problem | Previous time | Time Here | Lower Bound |
|---|---|---|---|
| classification/perceptron | $\tilde{O}(\varepsilon^{-2}M)$ [1] | $\tilde{O}(\varepsilon^{-2}(n+d))$ §II | $\Omega(\varepsilon^{-2}(n+d))$ §VII-A |
| min. enc. ball (MEB) | $\tilde{O}(\varepsilon^{-1/2}M)$ [8] | $\tilde{O}(\varepsilon^{-2}n+\varepsilon^{-1}d)$ §III-A | $\Omega(\varepsilon^{-1}n+\varepsilon^{-1}d)$ §VII-B |
| QP in the simplex | $O(\varepsilon^{-1}M)$ [9] | $\tilde{O}(\varepsilon^{-2}n+\varepsilon^{-1}d)$ §III-C | |
| Las Vegas versions | | additive $O(M)$ Cor II.11 | $\Omega(M)$ §VII-C |
| kernelized MEB and QP | | factors $O(s^4)$ or $O(q)$ §VI | |

Figure 1. Our results, except for semi-streaming and parallel

we minimize over $p \in \Delta$ and maximize over $x \in \mathbb{B}$, their optimization has not only $p$ but also $x$ in a unit simplex.

Their algorithm (and ours) relies on sampling based on $x$ and $p$, to estimate inner products $x^\top v$ or $p^\top w$ for vectors $v$ and $w$ that are rows or columns of $A$. For a vector $p \in \Delta$, this estimation is easily done by returning $w_i$ with probability $p_i$.

For vectors $x \in \mathbb{B}$, however, the natural estimation technique is to pick $i$ with probability $x_i^2$, and return $v_i/x_i$. The estimator from this $\ell_2$ *sample* is less well-behaved, since it is unbounded, and can have a high variance. While $\ell_2$ sampling has been used in streaming applications [7], it has not previously found applications in optimization due to this high variance problem.

Indeed, it might seem surprising that sublinearity is at all possible, given that the correct classifier might be determined by very few examples, as shown in figure 2. It thus seems necessary to go over all examples at least once, instead of looking at noisy estimates based on sampling.

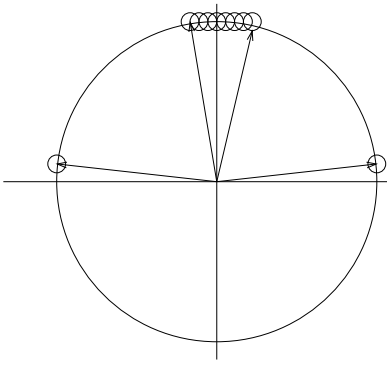

Figure 2. The optimum $x_*$ is determined by the vectors near the horizontal axis.

However, as we show, in our setting there is a version of the fundamental Multiplicative Weights (MW) technique that can cope with unbounded updates, and for which the variance of $\ell_2$-sampling is manageable. In our version of MW, the multiplier associated with a value $z$ is quadratic in $z$, in contrast to the more standard multiplier that is exponential in $z$; while the latter is a fundamental building block in approximate optimization algorithms, as discussed by Plotkin *et al.* [16], in our setting such exponential updates can lead to a very expensive $d^{\Omega(1)}$ iterations.

We analyze MW from the perspective of on-line optimization, and show that our version of MW has low expected expected regret given only that the random updates have the variance bounds provable for $\ell_2$ sampling. We also use another technique from on-line optimization, a gradient descent variant which is better suited for the ball.

For the special case of zero-sum games in which the entries are all non-negative (this is equivalent to packing and covering linear programs), Koufogiannakis and Young [17] give a sublinear-time algorithm which returns a *relative* approximation in time $\tilde{O}(\varepsilon^{-2}(n+d))$. Our lower bounds show that a similar relative approximation bound for sublinear algorithms is impossible for general classification, and hence general linear programming.

## II. Linear Classification and the Perceptron

**Please note: space limitations require that we omit the proofs of most of our results from this abstract.**

Before our algorithm, some reminders and further notation: $\Delta \subset \mathbb{R}^n$ is the unit simplex $\{p \in \mathbb{R}^n \mid p_i \geq 0, \sum_i p_i = 1\}$, $\mathbb{B} \subset \mathbb{R}^d$ is the Euclidean unit ball, and the unsubscripted $\|x\|$ denotes the Euclidean norm $\|x\|_2$. The $n$-vector, all of whose entries are one, is denoted by $\mathbf{1}_n$.

The $i$'th row of the input matrix $A$ is denoted $A_i$, although a vector is a column vector unless otherwise indicated. The $i$'th coordinate of vector $v$ is denoted $v(i)$. For a vector $v$, we let $v^2$ denote the vector whose coordinates have $v^2(i) \equiv v(i)^2$ for all $i$.

### A. The Sublinear Perceptron

Our sublinear perceptron algorithm is given in Figure 3. The algorithm maintains a vector $w_t \in \mathbb{R}^n$, with nonnegative coordinates, and also $p_t \in \Delta$, which is $w_t$ scaled to have unit $\ell_1$ norm. A vector $y_t \in \mathbb{R}^d$ is maintained also, and $x_t$ which is $y_t$ scaled to have Euclidean norm no larger than one. These normalizations are done on line 4.

1: Input: $\varepsilon > 0$, $A \in \mathbb{R}^{n\times d}$ with $A_i \in \mathbb{B}$ for $i \in [n]$.
2: Let $T \leftarrow 200^2\varepsilon^{-2}\log n$, $y_1 \leftarrow 0$, $w_1 \leftarrow \mathbf{1}_n$, $\eta \leftarrow \frac{1}{100}\sqrt{\frac{\log n}{T}}$.
3: **for** $t = 1$ to $T$ **do**
4: $\quad p_t \leftarrow \frac{w_t}{\|w_t\|_1}$, $x_t \leftarrow \frac{y_t}{\max\{1,\|y_t\|\}}$.
5: $\quad$ Choose $i_t \in [n]$ by $i_t \leftarrow i$ with prob. $p_t(i)$.
6: $\quad y_{t+1} \leftarrow y_t + \frac{1}{\sqrt{2T}}A_{i_t}$
7: $\quad$ Choose $j_t \in [d]$ by $j_t \leftarrow j$ with probability $x_t(j)^2/\|x_t\|^2$.
8: $\quad$ **for** $i \in [n]$ **do**
9: $\quad\quad \tilde{v}_t(i) \leftarrow A_i(j_t)\|x_t\|^2/x_t(j_t)$
10: $\quad\quad v_t(i) \leftarrow \mathrm{clip}(\tilde{v}_t(i), 1/\eta)$
11: $\quad\quad w_{t+1}(i) \leftarrow w_t(i)(1 - \eta v_t(i) + \eta^2 v_t(i)^2)$
12: $\quad$ **end for**
13: **end for**
14: **return** $\bar{x} = \frac{1}{T}\sum_t x_t$

Figure 3. Algorithm Sublinear Perceptron, a perceptron training algorithm

In lines 5 and 6, the algorithm is updating $y_t$ by adding a row of $A$ randomly chosen using $p_t$. This is a randomized version of *Online Gradient Descent* (OGD); due to the random choice of $i_t$, $A_{i_t}$ is an unbiased estimator of $p_t^\top A$, which is the gradient of $p_t^\top Ay$ with respect to $y$.

In lines 7 through 12, the algorithm is updating $w_t$ using a column $j_t$ of $A$ randomly chosen based on $x_t$, and also using the value $x_t(j_t)$. This is a version of the Multiplicative Weights (MW) technique for online optimization in the unit simplex, where $v_t$ is an unbiased estimator of $Ax_t$, the gradient of $p^\top Ax_t$ with respect to $p$.

Actually, $v_t$ is not unbiased, after the clip operation: for $z, V \in \mathbb{R}$, $\mathrm{clip}(z, V) \equiv \min\{V, \max\{-V, z\}\}$, and our analysis is helped by clipping the entries of $v_t$; we show that the resulting slight bias is not harmful.

As discussed in §I-A, the sampling used to choose $j_t$ (and update $p_t$) is $\ell_2$-*sampling*, and that for $i_t$, $\ell_1$-sampling. These techniques, which can be regarded as special cases of an $\ell_p$-sampling technique, for $p \in [1, \infty)$, yield unbiased estimators of vector dot products. It is important for us also that $\ell_2$-sampling has a variance bound here; in particular, for each relevant $i$ and $t$,

$$\mathbf{E}[v_t(i)^2] \le \|A_i\|^2\|x_t\|^2 \le 1. \tag{2}$$

First we note the running time.

**Theorem II.1.** *The sublinear perceptron takes $O(\varepsilon^{-2}\log n)$ iterations, with a total running time of $O(\varepsilon^{-2}(n+d)\log n)$.*

Next we analyze the output quality. The proof uses new tools from regret minimization and sampling that are the building blocks of most of our upper bound results.

Let us first state the MW algorithm used in all our algorithms.

**Definition II.2** (MW algorithm)**.** Consider a sequence of vectors $q_1, \ldots, q_T \in \mathbb{R}^n$. The *Multiplicative Weights* (MW) algorithm is as follows. Let $w_1 \leftarrow \mathbf{1}_n$, and for $t \ge 1$,

$$p_t \leftarrow w_t/\|w_t\|_1, \tag{3}$$

and for $0 < \eta \in \mathbb{R}$

$$w_{t+1}(i) \leftarrow w_t(i)(1 - \eta q_t(i) + \eta^2 q_t(i)^2), \tag{4}$$

The following is a key lemma, which proves a novel bound on the regret of the MW algorithm above, suitable for the case where the losses are random variables with bounded variance.

**Lemma II.3** (Variance MW Lemma)**.** *The MW algorithm satisfies*

$$\sum_{t\in[T]} p_t^\top q_t \quad \le \min_{i\in[n]} \sum_{t\in[T]} \max\{q_t(i), -\tfrac{1}{\eta}\} + \frac{\log n}{\eta} + \eta\sum_{t\in[T]} p_t^\top q_t^2.$$

The following three lemmas give concentration bounds on our random variables from their expectations. The first two are based on standard martingale analysis, and the last is a simple Markov application.

**Lemma II.4.** *For $\eta \le \sqrt{\frac{\log n}{10T}}$, with probability at least $1 - O(1/n)$,*

$$\max_i \sum_{t\in[T]} [v_t(i) - A_i x_t] \le 90\eta T.$$

**Lemma II.5.** *For $\eta \le \sqrt{\frac{\log n}{10T}}$, with probability at least $1 - O(1/n)$, it holds that $\left|\sum_{t\in[T]} A_{i_t}x_t - \sum_t p_t^\top v_t\right| \le 100\eta T$.*

**Lemma II.6.** *With probability at least $1 - \frac{1}{4}$, it holds that $\sum_t p_t^\top v_t^2 \le 8T$.*

**Theorem II.7** (Main Theorem)**.** *With probability $1/2$, the sublinear perceptron returns a solution $\bar{x}$ that is an $\varepsilon$-approximation.*

**Corollary II.8** (Dual solution)**.** *The vector $\bar{p} \equiv \sum_t e_{i_t}/T$ is, with probability $1/2$, an $O(\varepsilon)$-approximate dual solution.*

### *B. High Success Probability and Las Vegas*

Given two vectors $u, v \in \mathbb{B}$, we have seen that a single $\ell_2$-sample is an unbiased estimator of their inner product with variance at most one. Averaging $\frac{1}{\varepsilon^2}$ such samples reduces the variance to $\varepsilon^2$, which reduces the standard deviation to $\varepsilon$. Repeating $O(\log\frac{1}{\delta})$ such estimates, and taking the median, gives an estimator denoted $X_{\varepsilon,\delta}$, which satisfies, via a Chernoff bound:

$$\Pr[|X_{\varepsilon,\delta} - v^\top u| > \varepsilon] \le \delta$$

As an immediate corollary of this fact we obtain:

**Corollary II.9.** *There exists a randomized algorithm that with probability $1-\delta$, successfully determines whether a given hyperplane with normal vector $x \in \mathbb{B}$, together with an instance of linear classification and parameter $\sigma > 0$, is an $\varepsilon$-approximate solution. The algorithm runs in time $O(d + \frac{n}{\varepsilon^2}\log\frac{n}{\delta})$.*

Hence, we can amplify the success probability of Algorithm Sublinear Perceptron to $1-\delta$ for any $\delta > 0$ albeit incurring additional poly-log factors in running time:

**Corollary II.10** (High probability)**.** *There exists a randomized algorithm that with probability $1-\delta$ returns an $\varepsilon$-approximate solution to the linear classification problem, and runs in expected time $O(\frac{n+d}{\varepsilon^2}\log\frac{n}{\delta})$.*

It is also possible to obtain an algorithm that never errs:

**Corollary II.11** (Las Vegas Version)**.** *After $O(\varepsilon^{-2}\log n)$ iterations, the sublinear perceptron returns a solution that with probability $1/2$ can be verified in $O(M)$ time to be $\varepsilon$-approximate. Thus with expected $O(1)$ repetitions, and a total of expected $O(M+\varepsilon^{-2}(n+d)\log n)$ work, a verified $\varepsilon$-approximate solution can be found.*

### C. Implications in the PAC model

Consider the "separable" case of hyperplane learning, in which there exists a hyperplane classifying all data points correctly. It is well known that the concept class of hyperplanes in $d$ dimensions with margin $\sigma$ has effective dimension at most $\min\{d, \frac{1}{\sigma^2}\}+1$. Consider the case in which the margin is significant, i.e. $\frac{1}{\sigma^2} < d$. PAC learning theory implies that the number of examples needed to attain generalization error of $\delta$ is $O(\frac{1}{\sigma^2\delta})$.

Using the method of online to batch conversion (see [18]), and applying the online gradient decent algorithm, it is possible to obtain $\delta$ generalization error in time $O(\frac{d}{\sigma^2\delta})$ time, by going over the data once and performing a gradient step on each example.

Our algorithm improves upon this running time bound as follows: we use the sublinear perceptron to compute a $\sigma/2$-approximation to the best hyperplane over the test data, where the number of examples is taken to be $n = O(\frac{1}{\sigma^2\delta})$ (in order to obtain $\delta$ generalization error). As shown previously, the total running time amounts to $\tilde{O}(\frac{\frac{1}{\sigma^2\delta}+d}{\sigma^2}) = O(\frac{1}{\sigma^4\delta} + \frac{d}{\sigma^2})$.

This improves upon standard methods by a factor of $\tilde{O}(\sigma^2 d)$, which is always an improvement by our initial assumption on $\sigma$ and $d$.

## III. Strongly convex problems: MEB and SVM

### A. Minimum Enclosing Ball

In the Minimum Enclosing Ball problem the input consists of a matrix $A \in \mathbb{R}^{n\times d}$. The rows are interpreted as vectors and the problem is to find a vector $x \in \mathbb{R}^d$ such that

$$x_* \equiv \operatorname{argmin}_{x\in\mathbb{R}^d} \max_{i\in[n]} \|x - A_i\|^2$$

We further assume for this problem that all vectors $A_i$ have Euclidean norm at most one. Denote by $\sigma = \max_{i\in[n]}\|x - A_i\|^2$ the radius of the optimal ball, and we say that a solution is $\varepsilon$-approximate if the ball it generates has radius at most $\sigma + \varepsilon$.

As in the case of linear classification, to obtain tight running time bounds we use a primal-dual approach; this is combined with an approach of randomly skipping primal updates, to take advantage of the faster convergence of OGD for strongly convex functions. Due to space limitations we omit further description of the algorithm.

**Theorem III.1.** *Algorithm Sublinear MEB runs in $O(\frac{\log n}{\varepsilon^2})$ iterations, with a total expected running time of*

$$\tilde{O}\left(\frac{n}{\varepsilon^2} + \frac{d}{\varepsilon}\right),$$

*and with probability $1/2$, returns an $\varepsilon$-approximate solution.*

The following Corollary is a direct analogue of Corollary II.8.

**Corollary III.2** (Dual solution)**.** *The vector $\bar{p} \equiv \sum_t e_{i_t}/T$ is, with probability $1/2$, an $O(\varepsilon)$-approximate dual solution.*

### B. High Success Probability and Las Vegas

As for linear classification, we can amplify the success probability of Algorithm Sublinear MEB to $1-\delta$ for any $\delta > 0$ albeit incurring additional poly-log factors in running time.

**Corollary III.3** (MEB high probability)**.** *There exists a randomized algorithm that with probability $1-\delta$ returns an $\varepsilon$-approximate solution to the MEB problem, and runs in expected time $\tilde{O}(\frac{n}{\varepsilon^2}\log\frac{n}{\varepsilon\delta} + \frac{d}{\varepsilon}\log\frac{1}{\varepsilon})$. There is also a randomized algorithm that returns an $\varepsilon$-approximate solution in $\tilde{O}(M + \frac{n}{\varepsilon^2} + \frac{d}{\varepsilon})$ time.*

### C. Convex Quadratic Programming in the Simplex

We can extend our approach to problems of the form

$$\min_{p\in\Delta} p^\top b + p^\top A A^\top p, \tag{5}$$

where $b \in \mathbb{R}^n$, $A \in \mathbb{R}^{n\times d}$, and $\Delta$ is, as usual, the unit simplex in $\mathbb{R}^n$. As is well known, this problem includes the MEB problem, margin estimation as for hard margin support vector machines, the $L_2$-SVM variant of support vector machines, the problem of finding the shortest vector in a polytope, and others. We omit further discussion in this abstract.

Let $T \leftarrow \max\{T_\varepsilon(LRA), \frac{\log n}{\varepsilon^2}\}$ ,
$x_1 \leftarrow LRA(\text{initial})$, $w_1 \leftarrow \mathbf{1}_n$, $\eta \leftarrow \frac{1}{100}\sqrt{\frac{\log n}{T}}$.
**for** $t = 1$ to $T$ **do**
  **for** $i \in [n]$ **do**
    Let $v_t(i) \leftarrow$ **Sample**$(x_t, c_i)$
    $v_t(i) \leftarrow \text{clip}(\tilde{v}_t(i), 1/\eta)$
    $w_{t+1}(i) \leftarrow w_t(i)(1 - \eta v_t(i) + \eta^2 v_t(i)^2)$
  **end for**
  $p_t \leftarrow \frac{w_t}{\|w_t\|_1}$,
  Choose $i_t \in [n]$ by $i_t \leftarrow i$ with probability $p_t(i)$.
  $x_t \leftarrow LRA(x_{t-1}, c_{i_t})$
**end for**
**return** $\bar{x} = \frac{1}{T}\sum_t x_t$

Figure 4. Sublinear Primal-Dual Generic Algorithm

## IV. A Generic Sublinear Primal-Dual Algorithm

We note that our technique above can be applied more broadly to any constrained optimization problem for which low-regret algorithms exist and low-variance sampling can be applied efficiently; that is, consider the general problem with optimum $\sigma$:

$$\max_{x \in \mathcal{K}} \min_i c_i(x) = \sigma. \tag{6}$$

Suppose that for the set $\mathcal{K}$ and cost functions $c_i(x)$, there exists an iterative low regret algorithm, denoted $LRA$, with regret $R(T) = o(T)$. Let $T_\varepsilon(LRA)$ be the smallest $T$ such that $\frac{R(T)}{T} \leq \varepsilon$. We denote by $x_{t+1} \leftarrow LRA(x_t, c)$ an invocation of this algorithm, when at state $x_t \in \mathcal{K}$ and the cost function $c$ is observed.

Let **Sample**$(x, c)$ be a procedure that returns an unbiased estimate of $c(x)$ with variance at most one, that runs in constant time. Further assume $|c_i(x)| \leq 1$ for all $x \in K$ , $i \in [n]$.

Applying the techniques of section II we can obtain the following generic lemma.

**Lemma IV.1.** *The generic sublinear primal-dual algorithm returns a solution $x$ that with probability at least $\frac{1}{2}$ is an $\varepsilon$-approximate solution in $\max\{T_\varepsilon(LRA), \frac{\log n}{\varepsilon^2}\}$ iterations.*

High-probability results can be obtained using the same technique as for linear classification.

### A. More applications

The generic algorithm above can be used to derive the result of Grigoriadis and Khachiyan [15] on sublinear approximation of zero sum games with payoffs/losses bounded by one (up to poly-logarithmic factors in running time). A zero sum game can be cast as the following min-max optimization problem:

$$\min_{x \in \Delta_d} \max_{i \in \Delta_n} A_i x$$

That is, the constraints are inner products with the rows of the game matrix. This is exactly the same as the linear classification problem, but the vectors $x$ are taken from the convex set $\mathcal{K}$ which is the simplex - or the set of all mixed strategies of the column player.

A low regret algorithm for the simplex is the multiplicative weights algorithm, which attains regret $R(T) \leq 2\sqrt{T \log n}$. The procedure **Sample**$(x, A_i)$ to estimate the inner product $A_i x$ is much simpler than the one used for linear classification: we sample from the distribution $x$ and return $A_i(j)$ w.p. $x(j)$. This has correct expectation and variance bounded by one (in fact, the random variable is always bounded by one). Lemma IV.1 then implies:

**Corollary IV.2.** *The sublinear primal-dual algorithm applied to zero sum games returns a solution $x$ that with probability at least $\frac{1}{2}$ is an $\varepsilon$-approximate solution in $O(\frac{\log n}{\varepsilon^2})$ iterations and total time $\tilde{O}(\frac{n+d}{\varepsilon^2})$.*

Essentially any constrained optimization problem which has convex or linear constraints, and is over a simple convex body such as the ball or simplex, can be approximated in sublinear time using our method. The particular application to soft margin SVM, together with its practical significance, is explored in ongoing work with Nati Srebro.

## V. A Semi-Streaming Implementation

In order to achieve space that is sublinear in $d$, we cannot afford to output a solution vector. We instead output both the cost of the solution, and a set of indices $i_1, \ldots, i_t$ for which the solution is a linear combination (that we know) of $A_{i_1}, \ldots, A_{i_t}$. We note that all previous algorithms for these problems, even to achieve this notion of output, required $\Omega(d)$ space and/or $\Omega(nd)$ time, see, e.g., the references in [13].

We discuss the modifications to the sublinear primal-dual algorithm that need to be done for classification and minimum enclosing ball problems.

Our algorithm assumes it sees entire points at a time, i.e., it sees the entries of $A$ row at a time, though the rows may be ordered arbitrarily. It relies on two streaming results about a $d$-dimensional vector $x$ undergoing updates to its coordinates. We assume that each update is of the form $(i, z)$, where $i \in [d]$ is a coordinate of $x$ and $z \in \{-P, -P+1, \ldots, P\}$ indicates that $x_i \leftarrow x_i + z$. The first is an efficient $\ell_2$-sketching algorithm of Thorup and Zhang. This algorithm allows for $(1+\varepsilon)$-approximation of $\|x\|_2$ with high probability using 1-pass, $\tilde{O}(\varepsilon^{-2})$ space, and time proportinal to the length of the stream.

The second component is due to Monemizadeh and Woodruff [7]. We are given a stream of updates to a $d$-dimensional vector $x$, and want to output a random coordinate $I \in [d]$ for which for any $j \in [d]$, $\Pr[I = j] = \frac{|x_j|^2}{\|x\|_2^2}$. We also want the algorithm to return

the value $x_I$. Such an algorithm is called an exact augmented $\ell_2$-Sampler. As shown in [7], an augmented $\ell_2$-Sampler with $O(\log d)$ space, $\tilde{O}(1)$ passes, and running time $\tilde{O}(Q)$ exists, where $Q$ is the number of updates in the stream. This is what we use to $\ell_2$-sample from an iterate vector that we can only afford to represent implicitly.

We maintain the indices $i_t$ and $j_t$ used in all $\tilde{O}(\varepsilon^{-2})$ iterations of the primal dual algorithm. Notice that in a single iteration $t$ the same $\ell_2$-sample index $j_t$ can be used for all $n$ rows. While we cannot afford to remember the probabilities in the dual vector, we can store the values $\frac{\alpha_t}{x_t(j)}$, where $\alpha_t$ is a $(1 \pm \varepsilon)$-approximation of $\|x_t\|^2$ which can be obtained using the Thorup-Zhang sketch. We also need such an approximation to $\|x_t\|$ to appropriately weight the rows used to do $\ell_2$-sampling (see below). Since we see rows (i.e., points) of $A$ at a time, we can reconstruct the probability of each row in the dual vector on the fly in low space, and can use reservoir sampling to make the next choice of $i_t$. Then we use an augmented $\ell_2$-sampler to make the next choice of $j_t$, where we must $\ell_2$ sample from a weighted sum of rows indexed by $i_1, \ldots, i_t$ in low space. We can show that the algorithm remains correct given the per-iteration rounding of the updates $v_t(i)$ to relative error $\mu$, where $\mu$ is on the order of $\eta\epsilon/T$.

**Theorem V.1.** *There is an $\tilde{O}(\varepsilon^{-2})$-pass, $\tilde{O}(\varepsilon^{-2})$-space algorithm running in total time $\tilde{O}(\varepsilon^{-4}(n+d))$ which returns a list of $T = \tilde{O}(\varepsilon^{-2})$ row indices $i_1, \ldots, i_T$ which implicitly represent the normal vector to a hyperplane for $\varepsilon$-approximate classification, together with an additive-$\varepsilon$ approximation to the margin.*

For the MEB problem with high probability there are only $\tilde{O}(\varepsilon^{-1})$ different values of $i_t$ (i.e., updates to the primal vector). An important point is that we can get all $\tilde{O}(\varepsilon^{-1})$ $\ell_2$-samples independently from the same primal vector between changes to it by running the algorithm of [7] independently $\tilde{O}(\varepsilon^{-1})$ times in parallel.

We spend $\tilde{O}((n+d)\varepsilon^{-2})$ time per iteration, to reconstruct the dual vector and run the algorithm of [7] independently $\tilde{O}(\varepsilon^{-1})$ times on a stream of length $\tilde{O}(d\varepsilon^{-1})$ to do $\ell_2$-sampling).

**Theorem V.2.** *Given the norms of each row $A_i$, there is an $\tilde{O}(\varepsilon^{-1})$-pass, $\tilde{O}(\varepsilon^{-2})$-space algorithm running in total time $\tilde{O}(\varepsilon^{-3}(n+d))$ which returns a list of $T = \tilde{O}(\varepsilon^{-1})$ row indices $i_1, \ldots, i_T$ which implicitly represent the MEB center, together with an additive $\varepsilon$-approximation to the MEB radius.*

## VI. Kernelizing the Sublinear algorithms

An important generalization of linear classifiers is that of kernel-based linear predictors (see e.g. [19]). Let $\Psi : \mathbb{R}^d \mapsto \mathcal{H}$ be a mapping of feature vectors into a reproducing kernel Hilbert space. In this setting, we seek a non-linear classifier given by $h \in \mathcal{H}$ so as to maximize the margin:

$$\sigma \equiv \max_{h \in \mathcal{H}} \min_{i \in [n]} \langle h, \Psi(A_i) \rangle.$$

The kernels of interest are those for which we can compute inner products of the form $k(x,y) = \langle \Psi(x), \Psi(y) \rangle$ efficiently.

One popular kernel is the polynomial kernel, for which the corresponding Hilbert space is the set of polynomials over $\mathbb{R}^d$ of degree $q$. The mapping $\Psi$ for this kernel is given by

$$\forall S \subseteq [d] \ , \ |S| \leq q \ . \ \Psi(x)_S = \prod_{i \in S} x_i.$$

That is, all monomials of degree at most $q$. The kernel function in this case is given by $k(x,y) = (x^\top y)^q$. Another useful kernel is the Gaussian kernel $k(x,y) = \exp(-\frac{\|x-y\|^2}{2s^2})$, where $s$ is a parameter. The mapping here is defined by the kernel function (see [19] for more details).

In the full paper, we show that for both of these kernels have fast, unbiased, low variance estimators, based on $\ell_2$ sampling, and that such estimators can be leveraged to build fast estimators for the Hilbert-space inner products needed for a kernelized version of the the sublinear perceptron. The resulting Algorithm Sublinear Kernel has provably correct, sublinear performance, with the following bounds.

**Corollary VI.1.** *For the polynomial degree-q kernel, Algorithm Sublinear Kernel runs in time*

$$\tilde{O}(\frac{q(n+d)}{\varepsilon^2} + \min\{\frac{d \log q}{\varepsilon^4}, \frac{q}{\varepsilon^6}\}).$$

**Corollary VI.2.** *For the Gaussian kernel with parameter s, Algorithm Sublinear Kernel runs in time*

$$\tilde{O}(\frac{(n+d)}{s^4\varepsilon^2} + \min\{\frac{d}{\varepsilon^4}, \frac{1}{s^4\varepsilon^6}\}).$$

Analogously to Algorithm Sublinear Kernel, we can define the kernel version of strongly convex problems, including MEB. The kernelized version of MEB is particularly efficient, needing $\tilde{O}(\varepsilon^{-2}n + \varepsilon^{-1}d)$ time, for fixed $s$ or $q$.

## VII. Lower bounds

All of our lower bounds are information-theoretic, meaning that any successful algorithm must read at least some number of entries of the input matrix $A$. Clearly this also lower bounds the time complexity of the algorithm in the unit-cost RAM model.

Some of our arguments use the following meta-theorem. Consider a $p \times q$ matrix $A$, where $p$ is an even integer. Consider the following random process. Let $W \geq q$. Let $a = 1 - 1/W$, and let $e_j$ denote the $j$-th standard $q$-dimensional unit vector. For each $i \in [p/2]$, choose a random $j \in [q]$ uniformly, and set $A_{i+p/2} \leftarrow A_i \leftarrow ae_j + b(\mathbf{1}_q - e_j)$, where $b$ is

chosen so that $\|A_i\|_2 = 1$. We say that such an $A$ is a YES instance. With probability $1/2$, transform $A$ into a NO instance as follows: choose a random $i^* \in [p/2]$ uniformly, and if $A_{i^*} = ae_j + b(\mathbf{1}_q - e_j)$ for a particular $j^* \in [q]$, set $A_{i^*+p/2} \leftarrow -ae_{j^*} + b(\mathbf{1}_q - e_{j^*})$.

Suppose there is a randomized algorithm reading at most $s$ positions of $A$ which distinguishes YES and NO instances with probability $\geq 2/3$, where the probability is over the algorithm's coin tosses and this distribution $\mu$ on YES and NO instances. By averaging this implies a deterministic algorithm $Alg$ reading at most $s$ positions of $A$ and distinguishing YES and NO instances with probability $\geq 2/3$, where the probability is taken only over $\mu$. We show the following meta-theorem with a standard argument.

**Theorem VII.1.** *(Meta-theorem) For any such algorithm $Alg$, $s = \Omega(pq)$.*

### A. Classification

Recall that the margin $\sigma(A)$ of an $n \times d$ matrix $A$ is given by $\max_{x \in \mathbb{B}} \min_i A_i x$. Since we assume that $\|A_i\|_2 \leq 1$ for all $i$, we have that $\sigma(A) \leq 1$.

*1) Relative Error:* We start with a theorem for relative error algorithms.

**Theorem VII.2.** *Let $\kappa > 0$ be a sufficiently small constant. Let $\varepsilon$ and $\sigma(A)$ have $\sigma(A)^{-2}\varepsilon^{-1} \leq \kappa \min(n, d)$, $\sigma(A) \leq 1 - \varepsilon$, with $\varepsilon$ also bounded above by a sufficiently small constant. Also assume that $M \geq 2(n+d)$, that $n \geq 2$, and that $d \geq 3$. Then any randomized algorithm which, with probability at least $2/3$, outputs a number in the interval $[\sigma(A) - \varepsilon\sigma(A), \sigma(A)]$ must read*

$$\Omega(\min(M, \sigma(A)^{-2}\varepsilon^{-1}(n+d)))$$

*entries of $A$. This holds even if $\|A_i\|_2 = 1$ for all rows $A_i$.*

Notice that this yields a stronger theorem than assuming that both $n$ and $d$ are sufficiently large, since one of these values may be constant.

*2) Additive Error:* Here we give a lower bound for the additive error case. We give two different bounds, one when $\varepsilon < \sigma$, and one when $\varepsilon \geq \sigma$. Notice that $\sigma \geq 0$ since we may take the solution $x = \mathbf{0}_d$. The following is a corollary of Theorem VII.2.

**Corollary VII.3.** *Let $\kappa > 0$ be a sufficiently small constant. Let $\varepsilon, \sigma(A)$ be such that $\sigma(A)^{-1}\varepsilon^{-1} \leq \kappa \min(n, d)$ and $\sigma(A) \leq 1 - \varepsilon/\sigma(A)$, where $0 < \varepsilon \leq \kappa'\sigma$ for a sufficiently small constant $\kappa' > 0$. Also assume that $M \geq 2(n+d)$, $n \geq 2$, and $d \geq 3$. Then any randomized algorithm which, with probability at least $2/3$, outputs a number in the interval $[\sigma - \varepsilon, \sigma]$ must read*

$$\Omega(\min(M, \sigma^{-1}\varepsilon^{-1}(n+d)))$$

*entries of $A$. This holds even if $\|A_i\| = 1$ for all rows $A_i$.*

The following handles the case when $\varepsilon = \Omega(\sigma)$.

**Corollary VII.4.** *Let $\kappa > 0$ be a sufficiently small constant. Let $\varepsilon, \sigma(A)$ be such that $\varepsilon^{-2} \leq \kappa \min(n, d)$, $\sigma(A) + \varepsilon < \frac{1}{\sqrt{2}}$, and $\varepsilon = \Omega(\sigma)$. Also assume that $M \geq 2(n+d)$, $n \geq 2$, and $d \geq 3$. Then any randomized algorithm which, with probability at least $2/3$, outputs a number in the interval $[\sigma - \varepsilon, \sigma]$ must read*

$$\Omega(\min(M,\ \varepsilon^{-2}(n+d)))$$

*entries of $A$. This holds even if $\|A_i\| = 1$ for all rows $A_i$.*

### B. Minimum Enclosing Ball

**Theorem VII.5.** *Let $\kappa > 0$ be a sufficiently small constant. Assume $\varepsilon^{-1} \leq \kappa \min(n, d)$ and $\varepsilon$ is less than a sufficiently small constant. Also assume that $M \geq 2(n+d)$ and that $n \geq 2$. Then any randomized algorithm which, with probability at least $2/3$, outputs a number in the interval*

$$\left[\min_x \max_i \|x - A_i\|^2 - \varepsilon, \min_x \max_i \|x - A_i\|^2\right]$$

*must read*

$$\Omega(\min(M, \varepsilon^{-1}(n+d)))$$

*entries of $A$. This holds even if $\|A_i\| = 1$ for all rows $A_i$.*

### C. Las Vegas Algorithms

While our algorithms are Monte Carlo, meaning they err with small probability, it may be desirable to obtain Las Vegas algorithms, i.e., randomized algorithms that have low expected time but never err. We show this cannot be done in sublinear time.

**Theorem VII.6.** *For the classification and minimum enclosing ball problems, there is no Las Vegas algorithm that reads an expected $o(M)$ entries of its input matrix and solves the problem to within a one-sided additive error of at most $1/2$. This holds even if $\|A_i\| = 1$ for all rows $A_i$.*

## VIII. Concluding Remarks

We have described a general method for sublinear optimization of constrained convex programs, and showed applications to classical problems in machine learning such as linear classification and minimum enclosing ball obtaining improvements in leading-order terms over the state of the art. The application of our sublinear primal-dual algorithms to soft margin SVM and related convex problems is currently explored in ongoing work with Nati Srebro.

In all our running times the dimension $d$ can be replaced by the parameter $S$, which is the maximum over the input rows $A_i$ of the number of nonzero entries in $A_i$. Note that $d \geq S \geq M/n$. Here we require the assumption that entries of any given row can be recovered in $O(S)$ time, which is compatible with

keeping each row as a hash table or (up to a logarithmic factor in run-time) in sorted order.

*Acknowledgements:* We thank Nati Srebro and an anonymous referee for helpful comments on the relation between this work and PAC learning theory.

## REFERENCES

[1] A. Novikoff, "On convergence proofs on perceptrons." in *Proceedings of the Symposium on the Mathematical Theory of Automata, Vol XII*, 1962, pp. 615–622.

[2] M. L. Minsky and S. Papert, *Perceptrons: An introduction to computational geometry*. MIT press Cambridge, Mass, 1988.

[3] T. Bylander, "Learning linear threshold functions in the presence of classification noise," in *COLT '94: Proceedings of the Seventh Annual Conference on Computational Learning Theory*. New York, NY, USA: ACM, 1994, pp. 340–347.

[4] A. Blum, A. M. Frieze, R. Kannan, and S. Vempala, "A polynomial-time algorithm for learning noisy linear threshold functions," *Algorithmica*, vol. 22, no. 1/2, pp. 35–52, 1998.

[5] R. A. Servedio, "On PAC learning using winnow, perceptron, and a perceptron-like algorithm," in *COLT '99: Proceedings of the Twelfth Annual Conference on Computational Learning Theory*. New York, NY, USA: ACM, 1999, pp. 296–307.

[6] J. Dunagan and S. Vempala, "A simple polynomial-time rescaling algorithm for solving linear programs," in *STOC '04: Proceedings of the Thirty-Sixth Annual ACM Symposium on the Theory of Computing*. New York, NY, USA: ACM, 2004, pp. 315–320.

[7] M. Monemizadeh and D. Woodruff, "1-pass relative error $l_p$-sampling with applications," in *SODA '10: Proc. Twenty-First ACM-SIAM Symposium on Discrete Algorithms*, 2010.

[8] A. Saha and S. Vishwanathan, "Efficient approximation algorithms for minimum enclosing convex shapes," 2009, arXiv:0909.1062v2.

[9] M. Frank and P. Wolfe, "An algorithm for quadratic programming," *Naval Res. Logist. Quart.*, vol. 3, p. 95?110, 1956.

[10] K. L. Clarkson, "Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm," in *SODA '08: Proc. Nineteenth ACM-SIAM Symposium on Discrete Algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2008, pp. 922–931.

[11] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, "Graph distances in the data-stream model," *SIAM J. Comput.*, vol. 38, no. 5, pp. 1709–1727, 2008.

[12] S. Muthukrishnan, "Data streams: Algorithms and applications," *Foundations and Trends in Theoretical Computer Science*, vol. 1, no. 2, 2005.

[13] P. Agarwal and R. Sharathkumar, "Streaming algorithms for extent problems in high dimensions," in *SODA '10: Proc. Twenty-First ACM-SIAM Symposium on Discrete Algorithms*, 2010.

[14] H. Zarrabi-Zadeh and T. M. Chan, "A simple streaming algorithm for minimum enclosing balls," in *CCCG*, 2006.

[15] M. D. Grigoriadis and L. G. Khachiyan, "A sublinear-time randomized approximation algorithm for matrix games," *Operations Research Letters*, vol. 18, pp. 53–58, 1995.

[16] S. A. Plotkin, D. B. Shmoys, and E. Tardos, "Fast approximation algorithms for fractional packing and covering problems," in *SFCS '91: Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 1991, pp. 495–504.

[17] C. Koufogiannakis and N. E. Young, "Beating simplex for fractional packing and covering linear programs," in *FOCS*. IEEE Computer Society, 2007, pp. 494–504.

[18] N. Cesa-Bianchi, A. Conconi, and C. Gentile, "On the generalization ability of on-line learning algorithms," *IEEE Transactions on Information Theory*, vol. 50, no. 9, pp. 2050–2057, 2004.

[19] B. Schölkopf and A. J. Smola, "A short introduction to learning with kernels," pp. 41–64, 2003.