Efficient Protocols for Distributed Classification and Optimization

Hal Daumé III[1], Jeff M. Phillips[2], Avishek Saha[2], and Suresh Venkatasubramanian[2]

[1] University of Maryland, CP, MD 20742, USA
hal@umiacs.umd.edu
[2] University of Utah, SLC, UT 84112, USA
{jeffp,avishek,suresh}@cs.utah.edu

Abstract. A recent paper [1] proposes a general model for distributed learning that bounds the communication required for learning classifiers with ε error on linearly separable data adversarially distributed across nodes. In this work, we develop key improvements and extensions to this basic model. Our first result is a two-party *multiplicative-weight-update* based protocol that uses $O(d^2 \log 1/\varepsilon)$ words of communication to classify distributed data in arbitrary dimension d, ε-optimally. This extends to classification over k nodes with $O(kd^2 \log 1/\varepsilon)$ words of communication. Our proposed protocol is simple to implement and is considerably more efficient than baselines compared, as demonstrated by our empirical results.

In addition, we show how to solve fixed-dimensional and high-dimensional linear programming with small communication in a distributed setting where constraints may be distributed across nodes. Our techniques make use of a novel connection from multipass streaming, as well as adapting the multiplicative-weight-update framework more generally to a distributed setting.

1 Introduction

In recent years, distributed learning (learning from data spread across multiple locations) has witnessed a lot of research interest [2]. One of the major challenges in distributed learning is to minimize communication overhead between different parties, each possessing a disjoint subset of the data. Recent work [1] has proposed a distributed learning model that seeks to minimize communication (in a series of rounds) by carefully choosing the most informative data points at each node in each round. The authors present a number of general sampling based results as well as a specific two-way protocol that provides a logarithmic error bound on communication for the family of linear classifiers in $\mathbb{R}^2$. Most of their results pertain to two players but they propose basic (and as we will see, inefficient) extensions for multi-player scenarios. A distinguishing feature of this model is that it is *adversarial*. Except linear separability, no distributional or other assumptions are made on the data or how it is distributed across nodes.

In this paper, we develop this model substantially with new algorithmic ideas for solving learning problems. First, we extend the results on linear classification to arbitrary dimensions, in the process presenting a more general algorithm that does not rely on explicit geometric constructions. This approach exploits the multiplicative weight update (MWU) framework (specifically its use in boosting) and retains desirable theoretical guarantees – *data-size-independent* communication between nodes in order to classify data – while being simple to implement. Moreover, it easily extends to k-players, scaling only linearly in k, improving earlier results in two dimensions by a factor of k.

Motivated by the insight that MWU can be used to solve distributed learning problems, we propose approximate solutions for distributed semidefinite programming. In addition, we show how a *generic* multipass streaming algorithm for a problem can be made distributed, and apply this framework to solving linear programming in a distributed setting both exactly and approximately. Together, these results indicate that general optimization problems can be solved efficiently in our model. Exploiting the strong link between learning and optimization will then open the door to deploying many other learning tasks in the distributed setting with minimal communication.

Related work. Existing work in distributed learning mainly focuses on either inferring an accurate global classifier from multiple distributed sub-classifiers learned individually (at respective nodes) or on improving the efficiency of the overall learning protocol. The first line of work consists of techniques like *parameter mixing* [3, 4] or *averaging* [5] and classifier *voting* [6]. These approaches do admit convergence results but lack any useful bounds on the communication. Voting, on the other hand, has been shown [1] to yield suboptimal results on adversarially partitioned datasets. The goal of the second line of work is to make distributed algorithms scale to large datasets [7]; many of these works [8, 9] focus on MapReduce. [10] proposed a MapReduce based improved parallel stochastic gradient descent and more recently [11] improved the time complexity of γ-margin parallel algorithms from $\Omega(1/\gamma^2)$ to $O(1/\gamma)$.

Surprisingly absent in the above lines of work is the direct study of how to use communication sparingly in learning. And as [1] and this work demonstrates, intelligent interaction between nodes, communicating key data subsets not just its classification, can greatly reduce the necessary communication over existing approaches. On large distributed systems, communication has become a major bottleneck for many real-world problems; it accounts for a large percentage of total energy costs, and is the main reason that MapReduce algorithms are designed to minimize rounds (of communication). This strongly motivates the need to incorporate the study of this aspect of an algorithm directly, as presented and modeled in this paper.

Independently of this work[3], research by [12] considers very similar models to those of [1]. They also consider adversarially distributed data among k parties and provide algorithms to learn while minimizing the total communication between the parties. Like [1] the work of [12] presents both agnostic and non-agnostic results for generic settings, and shows improvements over sampling bounds in several specific settings including the d-dimensional linear classifier problem we consider here (also drawing inspiration from boosting). In addition, their work provides total communication bounds for decision lists and for proper and non-proper learning of parity functions. They also extend the model so as to preserve differential and distributional privacy while conserving total communication, as a resource, during the learning process.

[3] Preliminary versions of [12] and this work [13] were coordinated to be placed on the arXiv on the same day.

In contrast, this work identifies optimization as a key primitive underlying many learning tasks, and focuses on solving the underlying optimization problems as a way to provide general communication-friendly distributed learning methods. We introduce techniques that rely on multiplicative weight updates and multi-pass streaming algorithms. Our main contributions include translating these techniques into this distributed setting and using them to solve LPs (and SDPs) in addition to solving for d-dimensional linear separators.

2 Background

Here we revisit the basic model [1].

Model. We assume that there are k parties $P_1, P_2, \ldots P_k$. Each party P_i possesses a dataset D_i that no other party has access to, and each D_i may have both positive and negative examples. The goal is to classify the full dataset $D = \cup_i D_i$ correctly. We assume that there exists a perfect classifier h^* from a family of classifiers $\mathcal{H}$ with associated range space $(D, \mathcal{H})$ and bounded VC-dimension ν. We are willing to allow ε-classification error on D so that up to $\varepsilon|D|$ points in total are misclassified.

Each *word* of data (e.g., a single point or vector in $\mathbb{R}^d$ counts as $O(d)$ words) passed between any pair of parties is counted towards the total communication; this measure in words allows us to examine the cost of extending to d-dimensions, and allows us to consider communication in forms other than example points, but does not hinder us with precision issues required when counting bits. For instance, a protocol that broadcasts a message of M words (say M/d points in $\mathbb{R}^d$) from one node to the other $k-1$ players costs $O(kM)$ communication. The goal is to design a protocol with as little communication as possible. We assume an *adversarial* model of data distribution; in this setting we prepare for the worst, and allow some *adversary* to determine which player gets which subset of D.

Sampling bounds. Given D and a family of classifiers with bounded VC-dimension ν, a random sample from D of size

$$s_{\varepsilon,\nu} = O(\min\{(\nu/\varepsilon)\log(\nu/\varepsilon), \nu/\varepsilon^2\}) \qquad (1)$$

has at most ε-classification error on D with constant probability [14], as long as there exists a perfect classifier. Throughout this paper we will assume that a perfect classifier exists. This constant probability of success can be amplified to $1-\delta$ with an extra $O(\log(1/\delta))$ factor of samples.

Randomly partitioned distributions. Assume that for all $i \in [1,k]$, each party P_i has a dataset D_i drawn from the same distribution. That is, all datasets D_i are identically distributed. This case is much simpler than what the remainder of this paper will consider. Using (1), each D_i can be viewed as a sample from the full set $D = \cup_i D_i$, and with *no* communication each party P_i can faithfully estimate a classifier with error $O((\nu/|D_i|)\log(\nu|D_i|))$ [1].

Henceforth we will focus on *adversarially* distributed data.

One-way protocols. Consider a restricted setting where protocols are only able to send data from parties P_i (for $i \geq 2$) to P_1; a restricted form of *one-way communication*. We can again use (1) so that all parties P_i send a sample S_i of size $s_{\varepsilon,\nu}$ to P_1, and then P_1

constructs a global classifier on $\cup_{i=2}^{k} S_i$ with ε-classification error $\cup_{i=1}^{k} D_i$; this requires $O(dks_{\varepsilon,\nu})$ words of communication for points in $\mathbb{R}^d$.

For specific classifiers [1] we can do better. For thresholds and intervals one can learn a *zero*-error distributed classifier using constant amount of one-way communication. The same can be achieved for axis-aligned rectangles with $O(kd^2)$ words of communication. However, those authors show that hyperplanes in $\mathbb{R}^d$, for $d \geq 2$, require at least $\Omega(k/\varepsilon)$ one-way bits of communication to learn an ε-error distributed classifier.

Two-way protocols. Hereafter, we consider two-way protocols where any two players can communicate back and forth. It has been shown [1] that, in $\mathbb{R}^2$, a protocol can learn linear classifiers with at most ε-classification error using at most $O(k^2 \log 1/\varepsilon)$ communication. This protocol is deterministic and relies on a complicated pruning argument, whereby in each round, either an acceptable classifier is found, or a constant fraction more of some party's data is ensured to be classified correctly.

3 Improved Random Sampling for *k*-players

Our first contribution is an improved two-way k-player sampling-based protocol using *two-way* communication and the sampling result in (1). We designate party P_1 as a coordinator. P_1 gathers the size of each player's dataset D_i, simulates sampling from each player completely at random, and then reports back to each player the number of samples to be drawn by it, in $O(k)$ communication. Then each other party P_i selects $s_{\varepsilon,\nu}|D_i|/|D|$ random points (in expectation), and sends them to the coordinator. The union of this set satisfies the conditions of the result from (1) over $D = \cup_i D_i$ and yields the following result.

Theorem 1. *For any hypothesis family with VC-dimension ν for points in $\mathbb{R}^d$, there exists a two-way k-player protocol using $O(kd + d\min\{(\nu/\varepsilon)\log(\nu/\varepsilon), \nu/\varepsilon^2\})$ total words of communication that achieves ε-classification error, with constant probability.*

Using two-way communication, this type of result can be made even more general. Consider the case where each P_i's dataset arrives in a continuous stream; this is known as a *distributed data stream* [15]. Then applying results of [16], we can continually maintain a sufficient random sample at the coordinator of size s_ε (using an generalization of reservoir sampling) communicating $O((k + s_{\varepsilon,\nu})d \log |D|)$ words.

Theorem 2. *Let each of k parties have a stream of data points D_i where $D = \cup_i D_i$. For any hypothesis family with VC-dimension ν for points in $\mathbb{R}^d$, there exists a two-way k-player protocol using $O((k + \min\{(\nu/\varepsilon)\log(\nu/\varepsilon), \nu/\varepsilon^2\})\ d \log |D|)$ total words of communication that maintains ε-classification error, with constant probability.*

4 A Two-Party Protocol

In this section, we consider only two parties and refer to them as A and B. $A's$ data is labeled D_A and B's data is labeled D_B ($|D_B| = n$). Our protocol, summarized in Algorithm 1, is called WEIGHTEDSAMPLING. In each round, A sends a classifier h_A to B

and B responds back with a set of points R_B, constructed by sampling from a weighting on its points. After T rounds (for $T = O(\log(1/\varepsilon))$), we will show that by voting on the result from the set of T classifiers h_A will misclassify at most $\varepsilon|D_B|$ points from D_B while being perfect on D_A, and hence $\varepsilon|D_B| < \varepsilon|D_B \cup D_A| = \varepsilon|D|$, yielding a ε-optimal classifier as desired.

Algorithm 1 WEIGHTEDSAMPLING

```
Input: $D_A, D_B$, parameters: $0 < \varepsilon < 1$
Output: $h_{AB}$ (classifier with $\varepsilon$-error on $D_A \cup D_B$)
Init: $R_B = \{\}$; $w_i^0 = 1 \ \forall x_i \in D_B$;
for t = 1 ... $T = 5\log_2(1/\varepsilon)$ do
    ——— A's move ———
    $D_A = D_A \cup R_B$; $h_A^t := Learn(D_A)$; send $h_A^t$ to $B$;
    ——— B's move ———
    $R_B$ := MWU ($D_B$, $h_A^t$, 0.75, 0.2); send $R_B$ to $A$;
end for
$h_{AB} = \mathsf{Majority}(h_A^1, h_A^2, \ldots, h_A^T)$;
```

R_B can construct its points in two ways: a random sample and a deterministic sample. We will focus on the randomized version since it is more practical, although it has slightly worse bounds in the two-party case. Then we will also mention and analyze the deterministic version.

It remains to describe how B's points are weighted and updated, which dictates how B constructs the sample sent to A. Initially, they are all given a weight $w_1 = 1$. Then the re-weighting strategy (described in Algorithm 2) is an instance of the multiplicative weight update framework; with each new classifier h_A from A, party B increases all weights of misclassified points by a $(1+\rho)$ factor, and does not change the weight for correctly classified points. We will show $\rho = 0.75$ is sufficient. Intuitively, this ensures that consistently misclassified points eventually get weighted high enough that they are very likely to be chosen as examples to be communicated in future rounds. The deterministic variant simply replaces Line 7 of Algorithm 2 with the weighted variant [17] of the deterministic construction of R_B [18]; see details below.

Note that this is roughly similar in spirit to the heuristic protocol [1] that exchanged support points and was called ITERATIVESUPPORTS, which we will experimentally compare against. But the protocol proposed here is less rigid, and as we will demonstrate next, this allows for a much less nuanced analysis.

4.1 Analysis

Our analysis is based on the multiplicative weight update framework (and closely resembles boosting). First, we state a key structural lemma. Thereafter, we use this lemma for our main result. For ease of readability, we defer all proofs to the appendix.

As mentioned above (see (1)), after collecting a random sample S_ε of size $s_{\varepsilon,d} = O(\min\{(d/\varepsilon)\log(d/\varepsilon), d/\varepsilon^2\})$ drawn over the entire dataset $D \subset \mathbb{R}^d$, a linear classifier learned on S_ε is sufficient to provide ε-classification error on all of D with constant probability. There exist deterministic constructions for these samples S_ε still of size $s_{\varepsilon,\nu}$ [18] (and sometimes slightly smaller [19]); although they provide at most ε-classification error with probability 1, they, in general, run in time exponential in ν. Note that the VC-dimension of linear classifiers in $\mathbb{R}^d$ is $O(d)$, and these results still holds when the points are weighted and the sample is drawn (respectively constructed [17]) and error measured with respect to this weighting distribution. Thus B could send $s_{\varepsilon,d}$ points to A, and we would be done; but this is too expensive. We restate this result with a constant c, so that at most a c fraction of the weights of points are mis-classified (later we show that $c = 0.2$ is sufficient with our framework). Specifically, setting $\varepsilon = c$ and rephrasing the above results yields the following lemma.

Algorithm 2 MWU (D_B, h_A^t, ρ, c)

```
Input: $h_A^t, D_B$, parameters: $0 < \rho < 1, 0 < c < 1$
Output: $R_B$ (a set of $s_{c,d}$ points)
for all ($x_i \in D_B$) do
    if($h_A^t(x_i) \neq y_i$) then $w_i^{t+1} = w_i^t(1+\rho)$;
    if($h_A^t(x_i) == y_i$) then $w_i^{t+1} = w_i^t$;
end for
randomly sample $R_B$ from $D_B$ (according to $w^{t+1}$);
```

Lemma 1. *Let B have a weighted set of points D_B with weight function $w : D_B \to \mathbb{R}^+$. For any constant $c > 0$, party B can send a set $S_{c,d}$ of size $O(d)$ (where the constant depends on c) such that any linear classifier that correctly classifies all points in $S_{c,d}$ will misclassify points in D_B with a total weight at most $c \sum_{x \in D_B} w(x)$. The set $S_{c,d}$ can be constructed deterministically, or a weighted random sample from (D_B, w) succeeds with constant probability.*

We first state the bound using the deterministic construction of the set $S_{c,d}$, and then extend it to the more practical (from an implementation perspective) random sampling result, but with a slightly worse communication bound.

Theorem 3. *The deterministic version of two-party two-way* WEIGHTEDSAMPLING *for linear separators in $\mathbb{R}^d$ misclassifies at most $\varepsilon|D|$ points after $T = O(\log(1/\varepsilon))$ rounds using $O(d^2 \log(1/\varepsilon))$ words of communication.*

In order to use random sampling, as suggested in Algorithm 2, we need to address the probability of failure of our protocol. More specifically, the set $S_{c,d}$ in Lemma 1 is of size $O(d \log(1/\delta'))$ and a linear classifier with no error on $S_{c,d}$ misclassifies points in D_B with weight at most $c \sum_{x \in D_B} w(x)$, with probability at least $1 - \delta'$. We want this probability of failure to be a constant δ over the entire course of the protocol. Setting $\delta' = \delta/T$, and applying the union bound implies that the probability of failure at any point in the protocol is at most $\sum_{i=1}^T \delta' = \sum_{i=1}^T \delta/T = \delta$. This increases the communication cost of each round to $O(d^2 \log(1/\delta')) = O(d^2 \log(\log(1/\varepsilon)/\delta)) = O(d^2 \log\log(1/\varepsilon))$ words, with a constant δ probability of failure. Thus, random sampling in WEIGHTEDSAMPLING requires a total of $O(d^2 \log(1/\varepsilon) \log\log(1/\varepsilon))$ words of communication. We formalize below.

Theorem 4. *The randomized two-party two-way protocol* WEIGHTEDSAMPLING *for linear separators in $\mathbb{R}^d$ misclassifies at most $\varepsilon|D|$ points, with constant probability, after $T = O(\log(1/\varepsilon))$ rounds using $O(d^2 \log(1/\varepsilon)\log\log(1/\varepsilon))$ words of communication.*

5 k-Party Protocol

In Section 3 we described a simple protocol (Theorem 1) to learn a classifier with ε-error jointly among k parties using $O(kd + d\min\{\nu/\varepsilon\log(\nu/\varepsilon), \nu/\varepsilon^2\})$ words of total communication. We now combine this with the two-party protocol from Section 4 to obtain a k-player protocol for learning a joint classifier with error ε.

We fix an arbitrary node (say P_1) as the coordinator for the k-player protocol of Theorem 1. Then P_1 runs a version of the two-player protocol (from Section 4) from A's perspective and where players $P_2,\ldots,P_k$ serve jointly as the second player B. To do so, we follow the distributed sampling approach outlined in Theorem 1. Specifically, we fix a parameter c (set $c = 0.2$). Each other node reports the total weight $w(D_i)$ of their data to P_1, who then reports back to each node what fraction of the total data $w(D_i)/w(D)$ they own. Then each player sends the coordinator a random sample of size $s_{c,d}w(D_i)/w(D)$. Recall that we require $s_{c,d} = O(d\log\log(1/\varepsilon))$ in this case to account for probability of failure over all rounds. The union of these sets at P_1 satisfies the sampling condition in Lemma 1 for $\cup_{i=2}^k D_i$. P_1 computes a classifier on the union of its data and this joint sample and all previous joint samples, and sends the resulting classifier back to all the nodes. Sending this classifier to each party requires $O(kd)$ words of communication. The process repeats for $T = \log_2(1/\varepsilon)$ rounds.

Theorem 5. *The randomized k-party protocol for ε-error linear separators in $\mathbb{R}^d$ terminates in $T = O(\log(1/\varepsilon))$ rounds using $O((kd + d^2\log\log(1/\varepsilon))\log(1/\varepsilon))$ words of communication, and has a constant probability of failure.*

The random sampling algorithm required a sample of size $O(d\log\log(1/\varepsilon))$. However we can achieve a different communication trade-off using the deterministic construction where, in each round, each party P_i communicates a deterministically constructed set $S_{c,i}$ of size $O(d)$. The coordinator P_1 computes a classifier that correctly classifies points from all of these sets having at most $cw(D_i)$ weight of points misclassified in each D_i. The error is at most $cw(D_i)$ on each dataset D_i and so the error on all sets is at most $c\sum_{i=2}^k w(D_i) = cw(D)$. Again using $T = O(\log(1/\varepsilon))$ rounds we can achieve the following result.

Theorem 6. *The deterministic k-party protocol for ε-error linear separators in $\mathbb{R}^d$ terminates in $T = O(\log(1/\varepsilon))$ rounds using $O(kd^2\log(1/\varepsilon))$ words of communication.*

6 Experiments

In this section, we compare WEIGHTEDSAMPLING with the following baselines for 2-party and k-party protocols.

- NAIVE: sends all data from $(k-1)$ nodes to a coordinator node and then learns at the coordinator.

- VOTING: trains classifiers at each individual node and sends over the $(k-1)$ classifiers to a coordinator node. For any datapoint, the coordinator node predicts the label by taking a vote over all k classifiers.
- RAND: each of the $(k-1)$ nodes sends a random sample of size $s_{\varepsilon,d}$ to a coordinator node and then a classifier is learned at the coordinator node using all of its own data and the samples received.
- RANDEMP: cheaper version of RAND that uses a random sample of size $9d$ from each party each round; this value was chosen to make this baseline technique as favorable as possible.
- MAXMARG: ITERATIVESUPPORTS that selects informative points heuristically [1]. We do not compare with MEDIAN [1] as it is not applicable beyond two dimensions.
- MWU: WEIGHTEDSAMPLING that randomly samples points based on the distribution of the weights and runs for $5\log(1/\varepsilon)$ number of rounds (ref. Section 4).
- MWUEMP: a cheaper version of MWU which is terminated early if the training error has reached $\varepsilon|D|$.

For all these methods, SVM (from libSVM [20] library), with a linear kernel, was used as the underlying classifier. We report training accuracy and communication cost. The training accuracy is computed over the combined dataset D with an ε value of 0.05 (where applicable). The communication cost (in words) of all methods are reported as ratios with reference to MWUEMP as the base method. All numbers reported are averaged over 10 runs of the experiments; standard deviations are reported where appropriate. For MWU and MWUEMP, we use $\rho = 0.75$.

Communication cost computation. Each example point incurs a cost of $d+1$ (d words to describe its position in $\mathbb{R}^d$ and 1 word to describe its sign). Similarly, each linear classifier requires $d+1$ words of communication (d words to describe its direction and 1 word to describe its offset). Note that given our cost computation, for some datasets the cost of RAND, RANDEMP and MWU can exceed the cost of NAIVE (see, for example, *Cancer*).

Datasets. Six datasets, three each for two-party and four-party case, have been generated synthetically from mixture of Gaussians. Each Gaussian has been carefully seeded to generate different data partitions. For *Synthetic1*, *Synthetic2*, *Synthetic4*, *Synthetic5*, each node contains 5000 data points (2500 positive and 2500 negative) whereas for *Synthetic3* and *Synthetic6*, each node contains 8500 data points (4250 positive and 4250 negative) and all of these datapoints lie in 50 dimensions. Additionally, we investigate the performance of our protocols on real-world datasets. We use *Cancer* and *Mushroom* from the LibSVM data repository [20] as these datasets are linearly or almost linearly separable. This shows that although our protocols were designed for noiseless data they work well on noisy datasets too. However, when applied on noisy data, we do not guarantee the accuracy bounds that were claimed for noiseless datasets.

In Tables 1-2, we highlight (in bold) the protocol that performs the best. By best we mean that the method has the cheapest communication cost as well an accuracy that is more that $(1-\varepsilon)$ times the optimal, i.e., 95% for $\varepsilon = 0.05$. As will be frequently seen for VOTING, the communication cost is the cheapest but the accuracy is far from the desired ε-error specified, and in such circumstances we do not deem VOTING as the best method.

Table 1. Mean accuracy (Acc) and communication cost (Cost) required for synthetic datasets.

	Synthetic1		***Synthetic2***		***Synthetic3***	
	Acc	**Cost**	**Acc**	**Cost**	**Acc**	**Cost**
	2-party					
Naive	99.23 (0.0)	49.0	97.91 (0.0)	6.18	97.39 (0.0)	19.1
Voting	**95.00 (0.0)**	**0.01**	60.64 (0.0)	0.01	74.55 (0.0)	0.01
Rand	99.02 (0.0)	29.4	97.72 (0.0)	3.71	97.16 (0.0)	6.74
RandEmp	96.64 (0.1)	4.41	**95.13 (0.1)**	**0.56**	96.03 (0.1)	1.01
MaxMarg	96.39 (0.0)	4.26	93.76 (0.0)	6.18	73.62 (0.0)	19.1
Mwu	98.66 (0.1)	49.5	97.59 (0.1)	6.24	97.11 (0.1)	11.3
MwuEmp	95.00 (0.0)	1.00	95.17 (0.1)	1.00	**95.25 (0.2)**	**1.00**
	Synthetic4		***Synthetic5***		***Synthetic6***	
	4-party					
Naive	99.26 (0.0)	100	97.97 (0.0)	12.7	97.47 (0.0)	54.8
Voting	**95.00 (0.0)**	**0.01**	65.83 (0.0)	0.01	75.52 (0.0)	0.01
Rand	99.18 (0.0)	60.0	97.83 (0.0)	7.63	97.39 (0.0)	19.4
RandEmp	97.33 (0.1)	9.00	96.61 (0.1)	1.15	96.67 (0.1)	2.90
MaxMarg	95.95 (0.0)	0.82	93.94 (0.0)	15.2	75.05 (0.0)	80.2
Mwu	98.03 (0.2)	34.8	97.30 (0.1)	4.45	96.87 (0.1)	11.2
MwuEmp	95.11 (0.3)	1.00	**95.11 (0.2)**	**1.00**	**95.45 (0.2)**	**1.00**

6.1 Synthetic Results

Table 1 compares the performance metrics of the aforementioned protocols for *two*-parties. As can be seen, Voting performs the best for *Synthetic1* and RandEmp performs the best for *Synthetic2*. For *Synthetic3*, MwuEmp requires the least amount of communication to learn an ε-optimal distributed classifier. Note that, for *Synthetic2* and *Synthetic3*, both Voting and MaxMarg fail to produce a ε-optimal ($\varepsilon = 0.05$) classifier. MaxMarg exhibits this behavior despite incurring a communication cost that is as high as Naive (i.e., the accumulated cost of the support points become the same as the cost of Naive at which point we stop the algorithm).

In Table 1, most of the two-party results carry over to the multiparty case. Voting is the best for *Synthetic4* whereas MwuEmp is the best for *Synthetic5* and *Synthetic6*. As earlier, both Voting and MaxMarg do not yield 0.05-optimal classifiers for *Synthetic5* and *Synthetic6*.

Figure 1 (for two-party using *Synthetic1*) shows the communication costs (in *log-scale*) with variations in the number of data points per node and the dimension of the data. Note that we do not report the numbers for MaxMarg since MaxMarg takes a long time to finish. However, for *Synthetic1* the numbers for MaxMarg are similar to those of RandEmp and so their traces are similar. Note that in Figure 1, the cost of Naive increases as the number of dimensions increase. This is because the cost is multiplied by a factor of $(d+1)$, when expressed in words.

6.2 Real-world Results

Table 2 presents results for two and four-party protocols using real-world datasets. Other than two-party case for *Mushroom*, Voting performs best in all other cases.

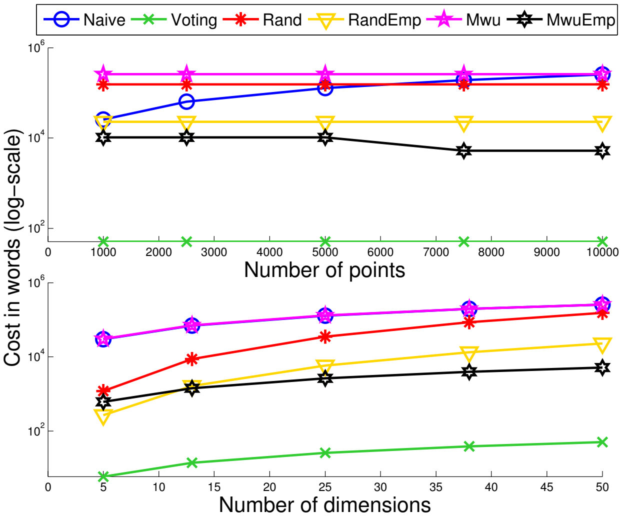

Fig. 1. Communication cost vs Size and Dimensionality for 2-party protocol.

However, note that VOTING does not yield a 0.05-optimal distributed classifier for *Mushroom* using two-party protocol.

The results for communication cost (in *log-scale*) versus data size and communication cost (in *log-scale*) versus dimensionality are provided in Figure 2 for two-party protocol using the *Mushroom* dataset. MWUEMP (denoted by the black line) is comparable to MAXMARG and cheaper than all other baselines (except VOTING).

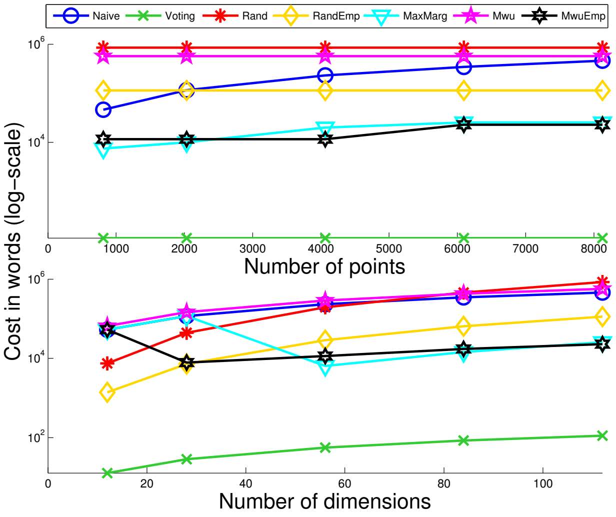

Fig. 2. Communication cost vs Size and Dimensionality for 2-party protocol.

Table 2. Results for *Cancer* ($|D| = 683$, $d = 10$) and *Mushroom* ($|D| = 8124$, $d = 112$).

	Cancer		***Mushroom***		***Cancer***		***Mushroom***	
	Acc	**Cost**	**Acc**	**Cost**	**Acc**	**Cost**	**Acc**	**Cost**
	2-party				4-party			
Naive	97.07 (0.0)	3.34	100.00 (0.0)	20.01	97.07 (0.0)	1.00	100.00 (0.0)	28.61
Voting	**97.36 (0.0)**	**0.01**	88.38 (0.0)	0.00	**97.36 (0.0)**	**0.03**	**95.67 (0.0)**	**0.01**
Rand	97.16 (0.1)	4.52	100.00 (1.1)	36.97	97.19 (0.1)	12.81	100.00 (0.6)	105.70
RandEmp	96.90 (0.2)	0.88	100.00 (0.0)	4.97	96.99 (0.1)	2.50	99.99 (0.0)	14.20
MaxMarg	96.78 (0.0)	0.22	100.00 (0.0)	1.11	96.78 (0.0)	0.56	100.00 (0.0)	2.34
Mwu	97.36 (0.2)	49.51	100.00 (0.0)	24.88	97.00 (0.2)	48.46	100.00 (0.1)	24.65
MwuEmp	96.87 (0.4)	1.00	**99.73 (0.5)**	**1.00**	96.97 (0.3)	1.00	98.86 (0.4)	1.00

Remarks. The goal of our experiments was to show that our protocols perform well, particularly on difficult or adversarially partitioned datasets. For easy datasets, any baseline technique can perform well. Indeed, Voting performs the best on *Synthetic1* and *Synthetic4* and RandEmp performs better than others on *Synthetic2*. For the remaining three cases on synthetic datasets, MwuEmp outperforms the other baselines. On real world data, Voting usually performs well. However, as we have seen, for some datasets Voting and MaxMarg fail to yield an ε-optimal classifier. In particular for *Mushroom*, using the two-party protocol, the accuracy achieved by Voting is far from ε-optimal. These results show that there exists scenarios where Voting and MaxMarg perform particularly worse and thus are not safe strategies.

7 Distributed Optimization

Thus far, we have focused on solving the binary classification problem in a distributed setting. Classification however is merely one kind of learning task, and one might ask whether other problems can be addressed using the MWU framework we describe. A useful insight here is that many learning tasks can be formulated as optimization problems in which the data act as constraints. For example, a simple linear SVM formulation has for each labeled point (x, y) the constraint $y(\langle w, x\rangle + b) \geq 1$.

Thus, a natural way to study a general class of learning tasks via optimization is as follows. Each player i has a set of constraints $C_i = \{f_{ij}(x) \geq 0\}$, and the goal is to solve the optimization $\min g(x)$ subject to the union of constraints $\cup_i C_i$. As earlier, our goal is to solve the above with minimum communication.

7.1 Optimization via Multiplicative Weight Updates

A first observation is that the MWU framework described in previous sections applies to distributed optimization. Consider the problem of solving a general LP of the form $\min g^\top x$, subject to $Ax \geq b$, $x \in P$, where P is a set of "soft" constraints (for example, $x \geq 0$) and $Ax \geq b$ are the "hard" constraints. Let $z^* = \min g^\top x^*$ be the optimal value of the LP, obtained at x^*. Then the multiplicative weight update method can be used to obtain a solution $\tilde{x}$ such that $z^* = g^\top \tilde{x}$ and all (hard) constraints are satisfied approximately, i.e $\forall i$, $A_i \tilde{x} \geq b_i - \varepsilon$, where $A_i x \geq b_i$ is one row of the constraint matrix. We call such a solution a *soft-ε-approximation* (to distinguish it from a traditional approximation in which all constraints would be satisfied *exactly* and the objective would be approximately achieved).

The standard protocol works as follows [21]. We assume that the optimal z^* has been guessed (this can be determined by binary search), and define the set of "soft" constraints to be $\mathcal{P} = P \cup \{x \mid g^\top x = z^*\}$. Typically, it is easy to check for feasibility in $\mathcal{P}$. We define a *width* parameter $\rho = \max\{\max_{i\in[n],x\in\mathcal{P}} A_i x - b_i, 1\}$. Initialize $m_i(0) = 0$. Then we run $T = O((\rho^2/\varepsilon^2)\ln n)$ iterations (with $t = 1, 2, \ldots, T$) of the following: (1) Set $p_i(t) = \exp(-\varepsilon m_i(t-1)/2)$, (2) Find feasible $x(t)$ in $\mathcal{P} \cup \{x \mid \sum_i p_i A_i x \geq \sum_i p_i b_i\}$, (3) $m_i(t) = m_i(t-1) + A_i x(t) - b_i$. At the end, we return $\bar{x} = (1/t)\sum_t x(t)$ as our soft-ε-approximation for the LP.

We now describe a two-party distributed protocol for linear programming adapted from this scheme. The protocol is asymmetric. Player A finds feasible values of x and player B maintains the weights m_i. Specifically, player A constructs a feasible set $\mathcal{P}$ consisting of the original feasible set P and all of its own constraints. As above, B initializes a weight vector m to all zeros, and then sends over the *single* constraint $\sum_i p_i A_i x \geq \sum_i p_i b_i$ to A. Player A then finds a feasible x using this constraint as well as $\mathcal{P}$ (solving a linear program) and then sends the resulting x back to B, who updates its weight vector m. Each round of communication requires $O(d)$ words, and there are $O((\rho^2/\varepsilon^2)\ln n)$ rounds of communication. Notice that this is exponentially better than merely sending over all constraints.

Theorem 7. *There is a 2-player distributed protocol that uses $O((d\rho^2/\varepsilon^2)\ln n)$ words of communication to compute a soft-ε-approximation for a linear program.*

A similar result applies for SDP (based on an existing primal MWU-based SDP algorithm [21]) as well as other optimizations for which the MWU applies, such as rank minimization [22], etc.

7.2 Optimization via Multi-Pass Streaming

We now present a different approach to distributed optimization. This approach introduces a novel reduction from *multipass streaming* to distributed communication. Given the extensive literature on streaming algorithms[23], this reduction is useful as a design strategy for algorithms in this model. Specifically, we show how fixed dimensional linear programming can be solved using this reduction.

A *streaming algorithm* [23] takes as input a sequence of items $x_1, \ldots x_n$. The algorithm is allowed working space that is *sublinear in n*, and is only allowed to look at each item once as it *streams* past. In *multipass* streaming, the algorithm may make more than one pass over the data, but is still limited to sublinear working space and a single look at each item in each pass. Lemma 2 shows that any (multipass) streaming algorithm can be used to build a multiparty distributed protocol.

Lemma 2. *A streaming algorithm for problem P that has s words of working storage and makes r passes over the data can be made into a k-player distributed algorithm for P that uses krs words of communication.*

Note that streaming algorithms often have $s = O(\text{poly} \log n)$ and $r = O(\log n)$, yielding (sublinear) $O(k \text{ poly} \log n)$ words of communication.

We can apply this lemma to get a distributed algorithm for fixed-dimensional linear programming[4]. This relies on an existing result [24]:

Theorem 8 ([24]). *Given n halfspaces in $\mathbb{R}^d$ (d constant), their lowest intersection point can be computed by a $O(1/\delta^{d-1})$-pass Las Vegas algorithm that uses $O((1/\delta^{O(1)})n^{\delta})$ space and runs in time $O((1/\delta^{O(1)})n^{1+\delta})$ with high probability, for any constant $\delta > 0$.*

Corollary 1. *There is a k-player algorithm for solving distributed linear programming that uses $O(k(1/\delta^{d+O(1)})n^{\delta})$ communication, for any constant $\delta > 0$.*

While the above streaming algorithm can be applied as a blackbox in Corollary 1, looking deeper into the streaming algorithm reveals room for improvement. As in the case of classification, suppose that we are permitted to violate an ε-fraction of the constraints. It turns out that the above streaming algorithm achieves its bounds by eliminating a fixed fraction of constraints in each space, and thus requires $\log_r n$ passes, where $r = n^{\Theta(\delta)}$. If we are allowed to violate an ε-fraction of constraints, we need only run the algorithm for $\log_r 1/\varepsilon$ passes, where r is now $O(1/\varepsilon^{\Theta(\delta)})$. This allows us to replace n in all terms by $1/\varepsilon$, resulting in an algorithm with communication *independent of n*.

Corollary 2. *There is a k-player algorithm for distributed linear programming that violates at most an ε-fraction of the constraints, and uses $O(k(1/\delta^{d+O(1)})(1/\varepsilon)^{\delta})$ communication, for any constant $\delta > 0$.*

8 Conclusion

In this work, we have proposed a simple and efficient protocol that learns an ε-optimal distributed classifier for hyperplanes in arbitrary dimensions. The protocol also gracefully extends to k-players. Our proposed technique WEIGHTEDSAMPLING relates to the MWU-based meta framework and we exploit this connection to extend WEIGHTEDSAMPLING for distributed convex optimization problems. This makes our protocol applicable to a wide variety of distributed learning problems that can be formulated as a convex optimization task over multiple distributed nodes.

References

[1] Daumé III, H., Phillips, J., Saha, A., Venkatasubramanian, S.: Protocols for learning classifiers on distributed data. In: AISTATS'12. (2012)
[2] Bekkerman, R., Bilenko, M., Langford, J., eds.: Scaling up Machine Learning: Parallel and Distributed Approaches. Cambridge University Press (2011)
[3] McDonald, R., Hall, K., Mann, G.: Distributed training strategies for the structured perceptron. In: NAACL HLT. (2010)

[4] Fixed-dimensional linear programming is the case of linear programming where the dimension is treated as a constant.

[4] Mann, G., McDonald, R., Mohri, M., Silberman, N., Walker, D.: Efficient large-scale distributed training of conditional maximum entropy models. In: NIPS. (2009)
[5] Collins, M.: Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In: EMNLP. (2002)
[6] Bauer, E., Kohavi, R.: An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. Machine Learning **36**(1-2) (1999)
[7] Dekel, O., Gilad-Bachrach, R., Shamir, O., Xiao, L.: Optimal distributed online prediction using mini-batches. arXiv:1012.1367 (2010)
[8] Chu, C.T., Kim, S.K., Lin, Y.A., Yu, Y., Bradski, G., Ng, A.Y., Olukotun, K.: Map-reduce for machine learning on multicore. In: NIPS. (2007)
[9] Teo, C.H., Vishwanthan, S., Smola, A.J., Le, Q.V.: Bundle methods for regularized risk minimization. J. Mach. Learn. Res. **11** (March 2010) 311–365
[10] Zinkevich, M., Weimer, M., Smola, A., Li, L.: Parallelized stochastic gradient descent. In: NIPS. (2010)
[11] Servedio, R.A., Long, P.: Algorithms and hardness results for parallel large margin learning. In: NIPS. (2011)
[12] Balcan, M.F., Blum, A., Fine, S., Mansour, Y.: Distributed learning, communication complexity and privacy. In: COLT'12 (to appear). (June 2012) arXiv:1204.3514.
[13] Daumé III, H., Phillips, J.M., Saha, A., Venkatasubramanian, S.: Efficient protocols for distributed classification and optimization. arXiv:1204.3523
[14] Anthony, M., Bartlett, P.L.: Neural Network Learning: Theoretical Foundations. Cambridge (2009)
[15] Cormode, G., Muthukrishnan, S., Yi, K.: Algorithms for distributed functional monitoring. In: SODA. (2008)
[16] Cormode, G., Muthukrishnan, S., Yi, K., Zhang, Q.: Optimal sampling from distributed streams. In: PODS. (2010)
[17] Matousek, J.: Approximations and optimal geometric divide-and-conquer. In: STOC. (1991)
[18] Chazelle, B.: The Discrepancy Method. Cambridge (2000)
[19] Matoušek, J.: Geometric Discrepancy. Springer (1999)
[20] Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM TIST **2**(3) (2011)
[21] Arora, S., Hazan, E., Kale, S.: Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In: FOCS. (2005)
[22] Meka, R., Jain, P., Caramanis, C., Dhillon, I.S.: Rank minimization via online learning. In: ICML. (2008)
[23] Muthukrishnan, S.: Data streams: algorithms and applications. Foundations and trends in theoretical computer science. Now Publishers (2005)
[24] Chan, T.M., Chen, E.Y.: Multi-pass geometric algorithms. Disc. & Comp. Geom. **37**(1) (2007) 79–102

Appendix

Proof of Theorem 3

Proof. At the start of each round t, let ϕ_t be the potential function given by the sum of weights of all points in that round. Initially, $\phi_1 = \sum_{x_i \in D_B} w_i = n$ since by definition for each point $x_i \in D_B$ we have $w_i = 1$.

Then in each round, A constructs a classifier h_A^t at B to correctly classify the set of points that accounts for at least $1 - c$ fraction of the total weight by Lemma 1. All other misclassified points are upweighted by $(1+\rho)$. Hence, for round $(t+1)$ we have $\phi^{t+1} \leq \phi^t\left((1-c)+c(1+\rho)\right) = \phi^t\left(1+c\rho\right) = n\left(1+c\rho\right)^t$.

Let us consider the weight of the points in the set $S \subset D_B$ that have been misclassified by a majority of the T classifiers (after the protocol ends). This implies every point in S has been misclassified *at least* $T/2$ number of times and *at most* T number of times. So the minimum weight of points in S is $(1+\rho)^{T/2}$ and the maximum weight is $(1+\rho)^T$.

Let n_i be the number of points in S that has weight $(1+\rho)^i$ where $i \in [T/2, T]$. The potential function value of S after T rounds is $\phi_S^T = \sum_{i=T/2}^{T} n_i(1+\rho)^i$. Our claim is that $\sum_{i=1}^{T} n_i = |S| \leq \varepsilon n$. Each of these at most $|S|$ points have a weight of at least $(1+\rho)^{T/2}$. Hence we have

$$\phi_S^T = \sum_{i=T/2}^{T} n_i(1+\rho)^i \geq (1+\rho)^{T/2} \sum_{i=T/2}^{T} n_i = (1+\rho)^{T/2}|S|.$$

Relating these two inequalities we obtain the following,

$$|S|(1+\rho)^{T/2} \leq \phi_S^T \leq \phi^T \leq n\left(1+c\rho\right)^T.$$

Hence (using $T = 5\log_2(1/\varepsilon)$)

$$|S| \leq n\left(\frac{(1+c\rho)}{(1+\rho)^{1/2}}\right)^{5\log_2(1/\varepsilon)} = n(1/\varepsilon)^{5\log_2\left(\frac{(1+c\rho)}{(1+\rho)^{1/2}}\right)}. \qquad (2)$$

Setting $c = 0.2$ and $\rho = 0.75$ we get $5\log_2\left((1+c\rho)/(1+\rho)^{1/2}\right) < -1$ and thus $|S| < n(1/\varepsilon)^{-1} < \varepsilon n$, as desired since $\varepsilon < 1$. Thus each round uses $O(d)$ points yielding a total communication of $O(d^2\log(1/\varepsilon))$ words.

Proof of Theorem 5

Proof. The correctness and bound of $T = O(\log(1/\varepsilon))$ rounds follows from Theorem 3, since, aside from the total weight gathering step, from party P_1's perspective it appears to run the protocol with some party B where B represents parties $P_2, P_3, \ldots, P_k$. The communication for P_1 to collect the samples from all parties is $O(kd + ds_{c,d}) = O(kd + d^2\log\log(1/\varepsilon))$. And it takes $O(dk)$ communication to return h_A to all $k-1$ other players. Hence the total communication over $T = O(\log(1/\varepsilon))$ rounds is $O((kd + d^2\log\log(1/\varepsilon))\log(1/\varepsilon))$ as claimed.

Proof of Lemma 2

Proof. First consider the case when $k = 2$. Consider a streaming algorithm S satisfying the conditions above. The simulation works by letting the first player A simulate the first half of S, and letting the second player B simulate the second half. Specifically, the first player A simulates the behavior of S on its input. When this simulation of S exhausts the input at A, A sends over the contents of the working store of S to B. B restarts S on its input using this working store as S's current state. When B has finished simulating S on its input, it sends the contents of the working storage back to A. This completes one pass of S, and used s words of communication. The process continues for r passes.

If there are k players $A_1, \ldots, A_k$ instead of two, then we fix an arbitrary ordering of the players. The first player simulates S on its input, and at completion passes the contents of the working store to the next one, and so on. Each pass now requires $O(ks)$ words of communication, and the result follows.