A Distributed Frank-Wolfe Algorithm for Communication-Efficient Sparse Learning

Aurélien Bellet, Yingyu Liang, Alireza Bagheri Garakani, Maria-Florina Balcan, Fei Sha

A Distributed Frank-Wolfe Algorithm for Communication-Efficient Sparse Learning

Aurélien Bellet* Yingyu Liang† Alireza Bagheri Garakani‡ Maria-Florina Balcan§
Fei Sha‡

Abstract

Learning sparse combinations is a frequent theme in machine learning. In this paper, we study its associated optimization problem in the distributed setting where the elements to be combined are not centrally located but spread over a network. We address the key challenges of balancing communication costs and optimization errors. To this end, we propose a distributed Frank-Wolfe (dFW) algorithm. We obtain theoretical guarantees on the optimization error ϵ and communication cost that do not depend on the total number of combining elements. We further show that the communication cost of dFW is optimal by deriving a lower-bound on the communication cost required to construct an ϵ-approximate solution. We validate our theoretical analysis with empirical studies on synthetic and real-world data, which demonstrate that dFW outperforms both baselines and competing methods. We also study the performance of dFW when the conditions of our analysis are relaxed, and show that dFW is fairly robust.

1 Introduction

Most machine learning algorithms are designed to be executed on a single computer that has access to the full training set. However, in recent years, large-scale data has become increasingly distributed across several machines (*nodes*) connected through a network, for storage purposes or because it is collected at different physical locations (as in cloud computing, mobile devices or wireless sensor networks [1, 2]). In these situations, communication between machines is often the main bottleneck and the naive approach of sending the entire dataset to a coordinator node is impractical. This distributed setting triggers many interesting research questions. From a fundamental perspective, studying the tradeoff between communication complexity [3, 4] and learning/optimization error has recently attracted some interest [5, 6, 7]. From a more practical view, a lot of research has gone into developing scalable algorithms with small communication and synchronization overhead (see for instance [8, 9, 10, 11, 12, 13, 14] and references therein).

In this paper, we present both theoretical and practical results for the problem of learning a sparse combination of elements (*atoms*) that are spread over a network. This has numerous applications in machine learning, such as LASSO regression [15], support vector machines [16], multiple kernel learning [17], ℓ_1-Adaboost [18] and sparse coding [19]. Formally, let $\boldsymbol{A} = [\boldsymbol{a}_1 \dots \boldsymbol{a}_n] \in \mathbb{R}^{d \times n}$ be a finite set of n d-dimensional atoms (one per column). Here, atoms broadly refer to data points, features, dictionary elements, basis functions, etc, depending on the application. We wish to find a weight vector $\boldsymbol{\alpha} \in \mathbb{R}^n$ (α_i giving the weight assigned to atom i) that minimizes some cost function $f(\boldsymbol{\alpha}) = g(\boldsymbol{A}\boldsymbol{\alpha})$ under a sparsity constraint:[1]

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \quad f(\boldsymbol{\alpha}) \quad \text{s.t.} \quad \|\boldsymbol{\alpha}\|_1 \leq \beta, \tag{1.1}$$

where the cost function $f(\boldsymbol{\alpha})$ is convex and continuously differentiable and $\beta > 0$.

We propose a distributed Frank-Wolfe (dFW) algorithm, a novel approach to solve such distributed sparse learning problems based on an adaptation of its centralized counterpart [20, 21, 22]. Intuitively, dFW is able to determine, in a communication-efficient way, which atoms are needed to reach an ϵ-approximate solution, and only those are broadcasted through the network. We also introduce an approximate variant of dFW that can be used to balance the amount of computation across the distributed machines to reduce synchronization costs. In terms of theoretical guarantees, we show

*LTCI UMR 5141, Télécom ParisTech & CNRS, aurelien.bellet@telecom-paristech.fr. Most of the work in this paper was carried out while the author was affiliated with the Department of Computer Science of the University of Southern California.

†Department of Computer Science, Princeton University, yingyul@cs.princeton.edu.

‡Department of Computer Science, University of Southern California, {bagherig,feisha}@usc.edu.

§School of Computer Science, Carnegie Mellon University, ninamf@cs.cmu.edu.

[1] Our results also hold for sparsity constraints other than the ℓ_1 norm, such as simplex constraints.

that the total amount of communication required by dFW is upper-bounded by a quantity that does not depend on n but only on ϵ, d and the topology of the network. We further establish that this dependency on ϵ and d is worst-case optimal for the family of problems of interest by proving a matching lower-bound of $\Omega(d/\epsilon)$ communication for any deterministic algorithm to construct an ϵ-approximation. Simple to implement, scalable and parameter-free, dFW can be applied to many distributed learning problems. We illustrate its practical performance on two learning tasks: LASSO regression with distributed features and Kernel SVM with distributed examples. Our main experimental findings are as follows: (i) dFW outperforms baseline strategies that select atoms using a local criterion, (ii) when the data and/or the solution is sparse, dFW requires less communication than ADMM, a popular competing method, and (iii) in a real-world distributed architecture, dFW achieves significant speedups over the centralized version, and is robust to random communication drops and asynchrony in the updates.

Outline Section 2 reviews the Frank-Wolfe algorithm in the centralized setting. In Section 3, we introduce our proposed dFW algorithm and an approximate variant of practical interest, as well as examples of applications. The theoretical analysis of dFW, including our matching communication lower bound, is presented in Section 4. Section 5 reviews some related work, and experimental results are presented in Section 6. Finally, we conclude in Section 7.

2 The Frank-Wolfe Algorithm

The Frank-Wolfe (FW) algorithm [20, 21, 22], also known as Conditional Gradient, is a procedure to solve convex constrained optimization problems of the form $\min_{\boldsymbol{\alpha}\in\mathcal{D}} f(\boldsymbol{\alpha})$ where f is convex and continuously differentiable, and $\mathcal{D}$ is a compact convex subset of any vector space (say $\mathbb{R}^n$). The basic FW algorithm is shown in Algorithm 1. At each iteration, it moves towards a feasible point $\boldsymbol{s}$ that minimizes a linearization of f at the current iterate. The stopping criterion uses a surrogate optimality gap that can be easily computed as a by-product of the algorithm. Let $\boldsymbol{\alpha}^*$ be an optimal solution. Theorem 2.1 shows that FW finds an ϵ-approximate solution in $O(1/\epsilon)$ iterations.

THEOREM 2.1. ([22]) *Let C_f be the curvature constant of f.*[2] *Algorithm 1 terminates after at most $6.75C_f/\epsilon$ iterations and outputs a feasible point $\tilde{\boldsymbol{\alpha}}$ which satisfies $f(\tilde{\boldsymbol{\alpha}}) - f(\boldsymbol{\alpha}^*) \leq \epsilon$.*

[2]The assumption of bounded C_f is closely related to that of L-Lipschitz continuity of ∇f with respect to an arbitrary chosen norm $\|\cdot\|$. In particular, we have $C_f \leq L\,\mathrm{diam}_{\|\cdot\|}(\mathcal{D})^2$ [22].

Algorithm 1 FW algorithm on general domain $\mathcal{D}$

1: Let $\boldsymbol{\alpha}^{(0)} \in \mathcal{D}$
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: $\quad \boldsymbol{s}^{(k)} = \arg\min_{\boldsymbol{s}\in\mathcal{D}} \left\langle \boldsymbol{s}, \nabla f(\boldsymbol{\alpha}^{(k)}) \right\rangle$
4: $\quad \boldsymbol{\alpha}^{(k+1)} = (1-\gamma)\boldsymbol{\alpha}^{(k)} + \gamma \boldsymbol{s}^{(k)}$, where $\gamma = \frac{2}{k+2}$ or obtained via line-search.
5: **end for**
6: **stopping criterion**: $\left\langle \boldsymbol{\alpha}^{(k)} - \boldsymbol{s}^{(k)}, \nabla f(\boldsymbol{\alpha}^{(k)}) \right\rangle \leq \epsilon$

Algorithm 2 Compute $\boldsymbol{s}^{(k)}$ for ℓ_1 / simplex

1: **if** ℓ_1 constraint
2: $\quad j^{(k)} = \arg\max_{j\in[n]} \left|\nabla f(\boldsymbol{\alpha}^{(k)})_j\right|$
3: $\quad \boldsymbol{s}^{(k)} = \mathrm{sgn}\left[-\nabla f(\boldsymbol{\alpha}^{(k)})_{j^{(k)}}\right] \beta \boldsymbol{e}^{j^{(k)}}$
4: **else if** simplex constraint
5: $\quad j^{(k)} = \arg\min_{j\in[n]} \nabla f(\boldsymbol{\alpha}^{(k)})_j$
6: $\quad \boldsymbol{s}^{(k)} = \boldsymbol{e}^{j^{(k)}}$

Solving the subproblems The key step in FW is to solve a linear minimization subproblem on $\mathcal{D}$ (step 3 in Algorithm 1). When the extreme points of $\mathcal{D}$ have a special structure (e.g., when they are sparse), it can be exploited to solve this subproblem efficiently. In particular, when $\mathcal{D}$ is an ℓ_1 norm constraint as in problem (1.1), a *single* coordinate (corresponding to the largest entry in the magnitude of the current gradient) is greedily added to the solution found so far (Algorithm 2). Taking $\boldsymbol{\alpha}^{(0)} = \mathbf{0}$, we get $\|\tilde{\boldsymbol{\alpha}}\|_0 \leq 6.75C_f/\epsilon$. Combining this observation with Theorem 2.1 shows that Algorithm 1 always finds an ϵ-approximate solution to problem (1.1) with $O(1/\epsilon)$ nonzero entries, which is sometimes called an ϵ-coreset of size $O(1/\epsilon)$ [23, 24]. It turns out that this is worst-case optimal (up to constant terms) for the family of problems (1.1), as evidenced by the derivation of a matching lower bound of $\Omega(1/\epsilon)$ for the sparsity of an ϵ-approximate solution [24].[3]

Another domain $\mathcal{D}$ of interest is the unit simplex $\Delta_n = \{\alpha \in \mathbb{R}^n : \boldsymbol{\alpha} \geq 0, \sum_i \alpha_i = 1\}$. The solution to the linear subproblem, shown in Algorithm 2, also ensures the sparsity of the iterates and thus leads to an ϵ-approximation with $O(1/\epsilon)$ nonzero entries, which is again worst-case optimal [21, 22].

For both domains, the number of nonzero entries depends only on ϵ, and not on n, which is particularly useful for tackling large-scale problems. These results constitute the basis for our design of a distributed algorithm with small communication overhead.

[3]Note that the dependency on ϵ can be improved to $O(n\log(1/\epsilon))$ for strongly convex functions by using away-steps (see e.g. [25]). However, when n is large, this improvement might not be desirable since it introduces a dependency on n.

Algorithm 3 Distributed Frank-Wolfe (dFW) algorithm for problem (1.1)

1: on all nodes: $\boldsymbol{\alpha}^{(0)} = \mathbf{0}$
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: on each node $v_i \in V$:
- $j_i^{(k)} = \arg\max_{j \in \mathcal{A}_i} \left| \nabla f(\boldsymbol{\alpha}^{(k)})_j \right|$ *// find largest entry of the local gradient in absolute value*
- $S_i^{(k)} = \sum_{j \in \mathcal{A}_i} \alpha_j^{(k)} \nabla f(\boldsymbol{\alpha}^{(k)})_j$ *// compute partial sum for stopping criterion*
- **broadcast** $g_i^{(k)} = \nabla f(\boldsymbol{\alpha}^{(k)})_{j_i^{(k)}}$ and $S_i^{(k)}$

4: on each node $v_i \in V$:
- $i^{(k)} = \arg\max_{i \in [N]} \left| g_i^{(k)} \right|$ *// compute index of node with largest overall gradient*
- **if** $i = i^{(k)}$ **then broadcast** $j^{(k)} = j_i^{(k)}$ and $\boldsymbol{a}_{j^{(k)}}$ **end if** *// send atom and index to other nodes*

5: on all nodes: $\boldsymbol{\alpha}^{(k+1)} = (1-\gamma)\boldsymbol{\alpha}^{(k)} + \gamma \operatorname{sgn}\left[-g_{i^{(k)}}^{(k)}\right] \beta \boldsymbol{e}^{j^{(k)}}$, where $\gamma = \frac{2}{k+2}$ or line-search
6: **end for**
7: **stopping criterion**: $\sum_{i=1}^{N} S_i^{(k)} + \beta \left| g_{i^{(k)}}^{(k)} \right| \leq \epsilon$

3 Distributed Frank-Wolfe (dFW) Algorithm

We now focus on the problem of solving (1.1) in the distributed setting. We consider a set of N local nodes $V = \{v_i\}_{i=1}^N$ which communicate according to an undirected connected graph $G = (V, E)$ where E is a set of M edges. An edge $(v_i, v_j) \in E$ indicates that v_i and v_j can communicate with each other. To simplify the analysis, we assume no latency in the network and synchronous updates — our experiments in Section 6 nonetheless suggest that these simplifying assumptions could be relaxed.

The atom matrix $\boldsymbol{A} \in \mathbb{R}^{d \times n}$ is partitioned column-wise across V. Formally, for $i \in [N]$, let the *local data* of v_i, denoted by $\mathcal{A}_i \subseteq [n]$, be the set of column indices associated with node v_i, where $\bigcup_i \mathcal{A}_i = [n]$ and $\mathcal{A}_i \bigcap \mathcal{A}_j = \emptyset$ for $i \neq j$. The *local gradient* of v_i is the gradient of f restricted to indices in $\mathcal{A}_i$. We measure the communication cost in number of real values transmitted.

3.1 Basic Algorithm Our communication-efficient procedure to solve (1.1) in the distributed setting is shown in Algorithm 3.[4] The algorithm is parameter-free and each iteration goes as follows: (i) each node first identifies the largest component of its local gradient (in absolute value) and broadcasts it to the other nodes (step 3), (ii) the node with the overall largest value broadcasts the corresponding atom, which will be needed by other nodes to compute their local gradients in subsequent iterations (step 4), and (iii) all nodes perform a FW update (step 5). We have the following result.

[4]Note that it can be straightforwardly adapted to deal with a simplex constraint based on Algorithm 2.

THEOREM 3.1. *Algorithm 3 terminates after $O(1/\epsilon)$ rounds and $O\left((Bd + NB)/\epsilon\right)$ total communication, where B is an upper bound on the cost of broadcasting a real number to all nodes in the network G. At termination, all nodes hold the same $\tilde{\boldsymbol{\alpha}}$ with optimality gap at most ϵ and the $O(1/\epsilon)$ atoms corresponding to its nonzero entries.*

Proof. We prove this by showing that dFW performs Frank-Wolfe updates and that the nodes have enough information to execute the algorithm. The key observation is that the local gradient only depends on local data and atoms received so far. Details can be found in the extended version of this paper [26].

Theorem 3.1 shows that dFW allows a trade-off between communication and optimization error without dependence on the total number of atoms, which makes the algorithm very appealing for large-scale problems. The communication cost depends only linearly on $1/\epsilon$, d and the network topology.

3.2 An Approximate Variant Besides its strong theoretical guarantees on optimization error and communication cost, dFW is very scalable, as the local computational cost of an iteration typically scales linearly in the number of atoms on the node. However, if these local costs are misaligned, the algorithm can suffer from significant wait time overhead due to the synchronized updates.

To address this issue, we propose an approximate variant of dFW which can be used to balance or reduce the effective number of atoms on each node. The algorithm is as follows. First, each node v_i clusters its local dataset $\mathcal{A}_i$ into m_i groups using the classic greedy m-center algorithm in [27], which repeatedly

Algorithm 4 GreedySelection($\mathcal{A}, \mathcal{C}, \Delta k$) // add Δk centers from $\mathcal{A}$ to $\mathcal{C}$

1: **repeat** Δk times
2: $\mathcal{C} = \mathcal{C} \cup \{\boldsymbol{a}_{j'}\}$, where $j' = \arg\max_{j \in \mathcal{A}} d(\boldsymbol{a}_j, \mathcal{C})$ with $d(\boldsymbol{a}_j, \mathcal{C}) = \min_{l \in \mathcal{C}} \|\boldsymbol{a}_j - \boldsymbol{a}_l\|_1$

Algorithm 5 Approximate dFW algorithm

1: on each node $v_i \in V$: $\boldsymbol{\alpha}^{(0)} = \mathbf{0}$; $\mathcal{C}_i = \text{GreedySelection}(\emptyset, m_i^{(0)})$
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: perform the same steps as in Algorithm 3, except that $j_i^{(k)}$ is selected from $\mathcal{C}_i$
4: on each node $v_i \in V$: $\mathcal{C}_i = \text{GreedySelection}(\mathcal{A}_i, \mathcal{C}_i, m_i^{(k+1)} - m_i^{(k)})$
5: **end for**

adds a new center that is farthest from the current set of centers (Algorithm 4). Then, dFW is simply run on the resulting centers. Intuitively, each original atom has a nearby center, thus selecting only from the set of centers is optimal up to small additive error that only leads to small error in the final solution. Optionally, we can also gradually add more centers so that the additive error scales as $O(1/k)$, in which case the error essentially does not affect the quality of the final solution. The details are given in Algorithm 5, and we have the following result (the proof is in the extended version [26]).

LEMMA 3.1. *Let $r^{opt}(\mathcal{A}, m)$ denote the optimal ℓ_1-radius of partitioning the local data indexed by $\mathcal{A}$ into m clusters, and let $r^{opt}(\boldsymbol{m}) := \max_i r^{opt}(\mathcal{A}_i, m_i)$. Let $G := \max_{\boldsymbol{\alpha}} \|\nabla g(\boldsymbol{A}\boldsymbol{\alpha})\|_\infty$. Then, Algorithm 5 computes a feasible solution with optimality gap at most $\epsilon + O(G r^{opt}(\boldsymbol{m}^0))$ after $O(1/\epsilon)$ iterations. Furthermore, if $r^{opt}(\boldsymbol{m}^{(k)}) = O(1/Gk)$, then the optimality gap is at most ϵ.*

Proof. By the analysis in [27], the maximum cluster radius on node v_i is at most $2r^{opt}(\mathcal{A}_i, m_i^{(k)})$ at time k, hence the maximum ℓ_1-radius over all clusters is at most $2r^{opt}(\boldsymbol{m}^{(k)})$. For any $j' \in [n]$, there exists $j \in \cup_i \mathcal{S}_i$ such that

$$\begin{aligned} |\nabla f(\boldsymbol{\alpha})_j - \nabla f(\boldsymbol{\alpha})_{j'}| &= |\boldsymbol{a}_j^{\mathrm{T}} \nabla g(\boldsymbol{A}\boldsymbol{\alpha}) - \boldsymbol{a}_{j'}^{\mathrm{T}} \nabla g(\boldsymbol{A}\boldsymbol{\alpha})| \\ &\leq 2r^{opt}(\boldsymbol{m}^{(k)}) G. \end{aligned}$$

Thus the selected atom is optimal up to a $2r^{opt}(\boldsymbol{m}^{(k)})G$ additive error. Then the first claim follows from the analysis in [28]. Similarly, when $r^{opt}(\boldsymbol{m}^{(k)}) = O(1/Gk)$, the additive error is $O(1/k)$. The second claim then follows from the analysis in [22].

Algorithm 5 has the potential to improve runtime over exact dFW in various practical situations. For instance, when dealing with heterogenous local nodes, v_i can pick m_i to be proportional to its computational power so as to reduce the overall waiting time. Another situation where this variant can be useful is when the distribution of atoms across nodes is highly unbalanced. We illustrate the practical benefits of this approximate variant on large-scale Kernel SVM in Section 6.

3.3 Illustrative Examples We give three examples of interesting problems that dFW can solve efficiently: LASSO regression, Kernel SVM and ℓ_1-Adaboost.

LASSO with distributed features The LASSO [15] is a linear regression method which aims at approximating a target vector $\boldsymbol{y} \in \mathbb{R}^d$ by a sparse combination of features:

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \quad \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\alpha}\|_2^2 \quad \text{s.t.} \quad \|\boldsymbol{\alpha}\|_1 \leq \beta, \tag{3.2}$$

where y_i and a_{ij} are the target value and j^{th} feature for training point i, respectively. Our dFW algorithm applies to problem (3.2) in the context of distributed features (each local node holds a subset of the features for all training points), as common in distributed sensor networks. It applies in a similar manner to sparse logistic regression [29] and, perhaps more interestingly, to Group-LASSO [30]. For the latter, suppose each group of features is located on the same node, for instance when combining features coming from the same source in multi-view learning or when using dummy variables that encode categorical features. In this case, each atom is a group of features and dFW selects a single group at each iteration (see [22] for the Frank-Wolfe update in this case).

SVM with distributed examples Let $\{\boldsymbol{z}_i = (\boldsymbol{x}_i, y_i)\}_{i=1}^n$ be a training sample with $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{\pm 1\}$. Let $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{x}') \rangle$ be a PSD kernel. The dual problem of L2-loss SVM with kernel k is as follows [31, 32]:

$$\min_{\boldsymbol{\alpha} \in \Delta_n} \quad \boldsymbol{\alpha}^{\mathrm{T}} \tilde{\boldsymbol{K}} \boldsymbol{\alpha}, \tag{3.3}$$

with $\tilde{\boldsymbol{K}} = [\tilde{k}(\boldsymbol{z}_i, \boldsymbol{z}_j)]_{i,j=1}^n$ where $\tilde{k}(\boldsymbol{z}_i, \boldsymbol{z}_j) = y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) + y_i y_j + \frac{\delta_{ij}}{C}$ is the "augmented kernel".

It is easy to see that $\tilde{k}(\boldsymbol{z}_i, \boldsymbol{z}_j) = \langle \tilde{\varphi}(\boldsymbol{z}_i), \tilde{\varphi}(\boldsymbol{z}_j) \rangle$ with $\tilde{\varphi}(\boldsymbol{z}_i) = [y_i \varphi(\boldsymbol{x}_i); y_i; \frac{1}{\sqrt{C}} \boldsymbol{e}_i]$. Thus $\tilde{\boldsymbol{K}} = \tilde{\boldsymbol{\Phi}}^{\mathrm{T}} \tilde{\boldsymbol{\Phi}}$ where $\tilde{\boldsymbol{\Phi}} = [\tilde{\varphi}(\boldsymbol{z}_1), \ldots, \tilde{\varphi}(\boldsymbol{z}_n)]^{\mathrm{T}}$ is the atom matrix.

Notice however that atoms may be of very high (or even infinite) dimension since they lie in kernel space. Fortunately, the gradient only depends on $\tilde{\boldsymbol{K}} \in \mathbb{R}^{n \times n}$, thus we can broadcast the training point $\boldsymbol{z}_i$ instead of $\tilde{\varphi}(\boldsymbol{z}_i)$ (assuming all nodes know the kernel function k). Note that dFW can be seen as a distributed version of Core Vector Machines [31], a large-scale SVM algorithm based on Frank-Wolfe, and that it applies in a similar manner to a direct multi-class L2-SVM [33] and to Structural SVM (see the recent work of [34]).

Boosting with distributed base classifiers Let $\mathcal{H} = \{h_j(\cdot) : \mathcal{X} \to \mathbb{R}, j = 1, \ldots, n\}$ be a class of base classifiers to be combined. Given a training sample $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^d$, let $\boldsymbol{A} \in \mathbb{R}^{d \times n}$ where $a_{ij} = y_i h_j(\boldsymbol{x}_i)$. We consider the following ℓ_1-Adaboost formulation [18]:

$$\min_{\boldsymbol{\alpha} \in \Delta_n} \quad \log \left(\frac{1}{d} \sum_{i=1}^{d} \exp \left(-\frac{(\boldsymbol{A}\boldsymbol{\alpha})_i}{T} \right) \right), \tag{3.4}$$

where $T > 0$ is a tuning parameter. Our algorithm thus applies to the case where the base classifiers are distributed across nodes. Note that the gradient of (3.4) is as follows:

$$\nabla f(x) = -\boldsymbol{w}^T \boldsymbol{A}, \quad \text{with } \boldsymbol{w} = \frac{\exp(-\boldsymbol{A}\boldsymbol{\alpha}/T)}{\sum_{i=1}^{d} \exp\left(-(\boldsymbol{A}\boldsymbol{\alpha})_i / T\right)}.$$

The FW update thus corresponds to adding the base classifier that performs best on the training sample weighted by $\boldsymbol{w}$, where $\boldsymbol{w}$ defines a distribution that favors points that are currently misclassified. This can be done efficiently even for a large (potentially infinite) family of classifiers when a weak learner for $\mathcal{H}$ is available.[5] In particular, when $\mathcal{H}$ is the class of decision stumps and the features are distributed, each node can call the weak learner on its local features to determine its best local base classifier.

4 Communication Complexity Analysis

In this section, we focus on communication complexity. We first give the communication cost of dFW depending on the network topology (Section 4.1). Then, we show that this communication complexity is worst-case optimal by proving a communication lower bound for the family of problems of interest (Section 4.2).

4.1 Communication Cost of dFW

We explicitly derive the communication cost of dFW for various types of network graphs (depicted in Figure 1).

[5] A weak learner for $\mathcal{H}$ returns the base classifier in $\mathcal{H}$ that performs best on the sample weighted by $\boldsymbol{w}$.

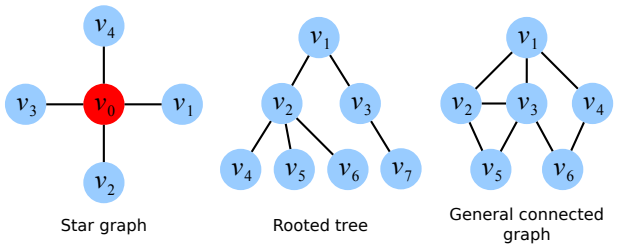


Figure 1: Illustration of network topologies.

Star network In a star network, there is a coordinator node v_0 connected to all local nodes $v_1, \ldots, v_N$. Broadcasting a number to all nodes has cost $B = N$, so the communication cost is $O\left((Nd + N^2)/\epsilon\right)$. This can be improved to $O(Nd/\epsilon)$: instead of broadcasting $g_i^{(k)}$ and $S_i^{(k)}$, we can send these quantities only to v_0 which selects the maximum $g_i^{(k)}$ and computes the sum of $S_i^{(k)}$.

Rooted tree Broadcasting on a tree costs $B = O(N)$, but this can be improved again by avoiding broadcast of $g_i^{(k)}$ and $S_i^{(k)}$. To select the maximum $g_i^{(k)}$, each node sends to its parent the maximum value among its local $g_i^{(k)}$ and the values received from its children. The root gets the maximum $g_i^{(k)}$ and can send it back to all the nodes. A similar trick works for computing the sum of $S_i^{(k)}$. Hence, only $j^{(k)}$ and the selected atom need to be broadcasted, and the total communication is $O(Nd/\epsilon)$.

General connected graph If the nodes can agree on a spanning tree, then this reduces to the previous case. We further consider the case when nodes operate without using any information beyond that in their local neighborhood and cannot agree on a spanning tree (this is known as the fully distributed setting [35]). In this case, broadcasting can be done through a message-passing procedure similar to [13]. The cost of broadcasting a number is $B = O(M)$, and thus the total communication cost is $O\left(M(N + d)/\epsilon\right)$.

4.2 Lower Bound on Communication Cost

In this section, we derive a lower bound on the communication required by any deterministic algorithm to construct an ϵ-approximation to problem (1.1) for either the ℓ_1-ball or the simplex constraint. We assume that at the end of the algorithm, at least one of the nodes has all the selected atoms. This is not a restrictive assumption since in many applications, knowledge of the selected atoms is required in order to use the learned model (for instance in kernel SVM).

Our proof is based on designing an optimization problem that meets the desired minimum amount of

communication. First, we consider a problem for which any ϵ-approximation solution must have $O(1/\epsilon)$ selected atoms. We split the data across two nodes v_1 and v_2 so that these atoms must be almost evenly split across the two nodes. Then, we show that for any fix dataset on one node, there are T different instances of the dataset on the other node, so that in any two such instances, the sets of selected atoms are different. Therefore, for any node to figure out the atom set, it needs $O(\log T)$ bits. The proof is completed by showing $\log T = \Omega(d/\epsilon)$.

Setup Consider the problem of minimizing $f(\boldsymbol{\alpha}) := \|\boldsymbol{A}\boldsymbol{\alpha}\|_2^2$ over the unit simplex Δ_d, where $\boldsymbol{A}$ is a $d \times d$ orthonormal matrix (i.e., $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}$). Suppose the first $d/2$ columns of $\boldsymbol{A}$ (denoted as $\boldsymbol{A}_1$) are on the node v_1 and the other half (denoted as $\boldsymbol{A}_2$) are on the node v_2. Restrict $\boldsymbol{A}$ to be block diagonal as follows: vectors in $\boldsymbol{A}_1$ only have non-zeros entries in coordinates $\{1, \ldots, d/2\}$ and vectors in $\boldsymbol{A}_2$ only have non-zeros entries in coordinates $\{d/2+1, \ldots, d\}$.

For any deterministic algorithm $\mathfrak{A}$, let $\mathfrak{A}(\boldsymbol{A}_1, \boldsymbol{A}_2)$ denote the set of atoms selected given input $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$, and let $f_{\mathfrak{A}}(\boldsymbol{A}_1, \boldsymbol{A}_2)$ denote the minimum value of $f(\boldsymbol{\alpha})$ when $\boldsymbol{\alpha} \in \Delta_d$ and $\boldsymbol{\alpha}$ has only non-zero entries on $\mathfrak{A}(\boldsymbol{A}_1, \boldsymbol{A}_2)$. We only consider deterministic algorithms that lead to provably good approximations: $\mathfrak{A}$ satisfies $f_{\mathfrak{A}}(\boldsymbol{A}_1, \boldsymbol{A}_2) < \epsilon + \text{OPT}$ for any $\boldsymbol{A}_1, \boldsymbol{A}_2$. Furthermore, at the end of the algorithm, at least one of the nodes has all the selected atoms.

In practice, bounded precision representations are used for real values. Study of communication in such case requires algebraic assumptions [4, 36]. For our problem, we allow the columns in $\boldsymbol{A}$ to be orthogonal and of unit length only approximately. More precisely, we say that two vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are near-orthogonal if $|\boldsymbol{v}_1^\top \boldsymbol{v}_2| \leq \frac{\epsilon}{4d^2}$, and a vector $\boldsymbol{v}$ has near-unit length if $|\boldsymbol{v}^\top \boldsymbol{v} - 1| \leq \frac{\epsilon}{4d^2}$. The notion of subspace is then defined based on near-orthogonality and near-unit length. We make the following assumption about the representation precision of vectors in subspaces.

ASSUMPTION 4.1. *There exists a constant $\kappa > 1$ s.t. for any t-dimensional subspace, $t \leq d$, the number of different d-dimensional near-unit vectors in the subspace is $\Theta(\kappa^{t-1})$.*

Analysis Let $\epsilon \in (0, 1/6)$ and $d = \frac{1}{6\epsilon}$ (for simplicity, suppose $d/4$ is an integer). We first show that the atoms in any good approximation solution must be almost evenly split across the two nodes.

CLAIM 4.1. *Any $\hat{\boldsymbol{\alpha}}$ with $f(\hat{\boldsymbol{\alpha}}) < \epsilon + \text{OPT}$ has more than $3d/4$ non-zero entries. Consequently, $|\mathfrak{A}(\boldsymbol{A}_p, \boldsymbol{A}_q) \cap \boldsymbol{A}_p| > d/4$ for $\{p, q\} = \{1, 2\}$.*

Next, we show that for any fix dataset on one node, there are many different instances of the dataset on the other node, so that any two different instances lead to different selected atoms.

CLAIM 4.2. *For any instance of $\boldsymbol{A}_1$, there exist $T = \kappa^{\Omega(d/\epsilon)}/(d/e)^{O(d)}$ different instances of $\boldsymbol{A}_2$ (denoted as $\{\boldsymbol{A}_2^i\}_{i=1}^T$) such that the set of atoms selected from $\boldsymbol{A}_2^i$ are different for all i, i.e., for any $0 \leq i \neq j \leq T$, $\mathfrak{A}(\boldsymbol{A}_1, \boldsymbol{A}_2^i) \cap \boldsymbol{A}_2^i \neq \mathfrak{A}(\boldsymbol{A}_1, \boldsymbol{A}_2^j) \cap \boldsymbol{A}_2^j$.*

Due to lack of space, the proofs of Claim 4.1 and 4.2 can be found in the extended version of the paper [26]. Distinguishing between the T instances requires communication, resulting in our lower bound.

THEOREM 4.1. *Suppose a deterministic algorithm $\mathfrak{A}$ satisfies $f_{\mathfrak{A}}(\boldsymbol{A}_1, \boldsymbol{A}_2) < \epsilon + \text{OPT}$ for any $\boldsymbol{A}_1, \boldsymbol{A}_2$. Under Assumption 4.1, the communication cost of $\mathfrak{A}$ is lower-bounded by $\Omega(\frac{d}{\epsilon}\log\kappa - d\log d)$ bits.*

Proof. Suppose the algorithm uses less than $\log T$ bits. At the end of the algorithm, if node v_1 has the selected atoms, then v_1 must have determined these atoms based on $\boldsymbol{A}_1$ and less than $\log T$ bits. By Claim 4.2, there are T different instances of $\boldsymbol{A}_2$ with different $\mathfrak{A}(\boldsymbol{A}_1, \boldsymbol{A}_2) \cap \boldsymbol{A}_2$. For at least two of these instances, v_1 will output the same set of atoms, which is contradictory. A similar argument holds if node v_2 has the selected atoms. Therefore, the algorithm uses at least $\log T$ bits.

Remarks The same lower bound holds for ℓ_1-constrained problems by reduction to the simplex case. The analysis also extends to networks with more than two nodes, in which case the diameter of the network is a multiplicative factor in the bound. Details can be found in the extended version of the paper [26].

The upper bound given by dFW in Theorem 3.1 matches the lower bound of Theorem 4.1 in its dependency on ϵ and d. This leads to the important result that the communication cost of dFW is worst-case optimal in these two quantities.

5 Related Work

In this section, we review some related work in distributed optimization.

Distributed examples An important body of work considers the distributed minimization of a function $f(\boldsymbol{\alpha}) = \sum_{i=1}^N f_i(\boldsymbol{\alpha})$ where f_i is convex and only known to node i. This typically corresponds to the risk minimization setting with distributed training examples, where f_i gives the loss for the data points located on node i. These methods are based on subgradient descent (SGD) ([11, 12] and references therein), the

Alternating Direction Methods of Multipliers (ADMM) [10, 37] and Stochastic Dual Coordinate Ascent (SDCA) [14]. Their rate of convergence (and thus their communication cost) for general convex functions is in $O(1/\epsilon)$ or $O(1/\epsilon^2)$, sometimes with a dependency on n (the total number of data points), which is undesirable for large-scale problems. The above algorithms focus on theoretical guarantees for the optimization error and pay less attention to minimizing the communication overhead. Overall, it is unclear whether any of them is optimal in the communication complexity sense.

Note that for Kernel SVM (Section 3.3), none of the above methods can be applied efficiently, either because they rely on the primal (GD, ADMM) or because they require broadcasting at each iteration a number of data points that is at least the number of nodes in the network (SDCA). In contrast, dFW only needs to broadcast a single point per iteration and still converges in $O(1/\epsilon)$, with no dependency on n. We illustrate its performance on this task in the experiments.

Distributed features In the case of distributed features, ADMM is often the method of choice due to its wide applicability and good practical performance [10].[6] Given N nodes, the parameter vector $\boldsymbol{\alpha}$ is partitioned as $\boldsymbol{\alpha} = [\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_N]$ with $\boldsymbol{\alpha}_i \in \mathbb{R}^{n_i}$, where $\sum_{i=1}^N n_i = n$. The matrix $\boldsymbol{A}$ is partitioned likewise as $\boldsymbol{A} = [\boldsymbol{A}_1 \ldots \boldsymbol{A}_N]$ and the optimization problem has the form

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \quad g\left(\sum_{i=1}^N \boldsymbol{A}_i \boldsymbol{\alpha}_i - \boldsymbol{y}\right) + \lambda R(\boldsymbol{\alpha}),$$

where the loss g is convex, the regularizer R is separable across nodes and convex, and $\lambda > 0$. Each node v_i iteratively solves a local subproblem and sends its local prediction $\boldsymbol{A}_i \boldsymbol{\alpha}_i$ to a coordinator node, which then broadcasts the current global prediction $\sum_{i=1}^N \boldsymbol{A}_i \boldsymbol{\alpha}_i$. ADMM can be slow to converge to high accuracy, but typically converges to modest accuracy within a few tens of iterations [10].

Both ADMM and dFW can deal with LASSO regression with distributed features. When features are dense, dFW and ADMM have a similar iteration cost but dFW is typically slower to converge as it only adds one feature to the solution at each iteration. On the other hand, the case of sparse features creates an interesting tradeoff between the number of iterations and the communication cost per iteration. Indeed, an iteration of dFW becomes much cheaper in communication than an iteration of ADMM since local/global predictions are typically dense. This tradeoff is studied in the experiments.

[6] Note that to the best of our knowledge, no convergence rate is known for ADMM with distributed features.

6 Experiments

We have shown that dFW enjoys strong theoretical guarantees on the communication cost. In this section, we validate our analysis with a series of experiments on two tasks: LASSO regression with distributed features, and Kernel SVM with distributed examples.

6.1 Summary of Main Results Our main results are as follows:

1. dFW decreases the objective better than two baselines that select atoms locally.

2. When data and/or the solution are sparse, dFW requires less communication than distributed ADMM [10], a popular competing method for distributed optimization.

3. dFW and its approximate variant perform well in a real-world distributed architecture.

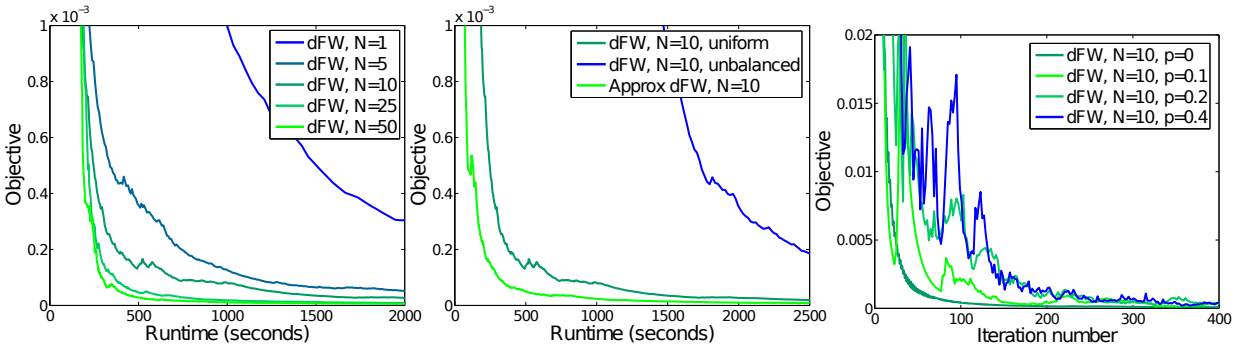
4. dFW is fairly robust to random communication drops and asynchronous updates.

Due to the lack of space, detailed experiments for 1-2 on both synthetic and real data can be found in the extended version of the paper [26]. In the following, we describe our results for 3-4.

6.2 Large-scale Distributed Experiments In this section, we apply dFW to kernel SVM (see Section 3.3) in real-world distributed conditions. Note that dFW is computationally very efficient if implemented carefully. In particular, the local gradient of a node v_i can be recursively updated using only the kernel values between its n_i points and the point received at the previous iteration. Thus the algorithm only needs $O(n_i)$ memory and the computational cost of an iteration is $O(n_i)$. We implemented dFW (without line-search) in C++ with openMPI. Our code is freely available under GNU/GPL license.[7] We experiment with a fully connected network with $N \in \{1, 5, 10, 25, 50\}$ nodes which communicate over a 56.6-gigabit infrastructure. Each node is a single 2.4GHz CPU core of a separate host. We use a speech dataset of 8.7M examples with 120 features and 42 class labels (the task is majority class versus rest).[8] In all experiments, we use the RBF kernel and set its bandwidth based on the averaged distance among examples. The parameter C of SVM is set to 100.

[7] `https://mloss.org/software/view/593/`

[8] The data is from IARPA Babel Program, Cantonese language collection release IARPA-babel101b-v0.4c limited data pack.



(a) Exact dFW on ideal distribution (b) Approx. dFW to balance costs (c) dFW under asynchrony

Figure 2: Empirical evaluation of our theoretical analysis on distributed Kernel SVM (best seen in color).

Exact dFW We first investigate how dFW scales with the number of nodes. For this experiment, training points are assigned to nodes uniformly at random. Figure 2(a) shows the runtime results for different values of N. We obtain near-linear speedup, showing that synchronization costs are not a significant issue when the local computation is well-balanced across nodes.

Approximate variant When the local costs are misaligned, the exact dFW will suffer from large wait time. As we have seen in Section 3.2, we can use the approximate variant of dFW to balance these costs. To illustrate this, we set $N = 10$ and assign about 50% of the data to a single node while other nodes share the rest uniformly at random. We use the greedy clustering algorithm on the large node to generate a number of centers approximately equal to the number of atoms on the other nodes, thus balancing local computation.[9] Figure 2(b) shows that the approximate dFW reduces the runtime dramatically over exact dFW on the unbalanced distribution. As often the case in large-scale datasets, training examples cluster well, so the additive error suffered is negligible in this case.

Asynchronous updates Another way to reduce waiting times and synchronization costs is to use asynchronous updates. To simulate this, we randomly drop the communication messages with some probability $p > 0$. This means that at any given iteration: (i) the selected atom may be suboptimal, and (ii) some nodes may not update their iterate. As a result, the iterates across nodes are not synchronized anymore. Figure 2(c) shows the objective value averaged over all the nodes at each iteration. We can see that dFW is robust to this asynchronous setting and converges properly (albeit a bit slower), even with 40% random communication drops. This suggests that an asynchronous version of dFW could lead to significant speedups in practice. A theoretical analysis of the algorithm in this challenging setting is an exciting direction for future work.

[9] For kernel SVM with RBF kernel, the difference between gradient entries of f can be directly bounded by performing clustering in the augmented kernel space, because $|\nabla f(\boldsymbol{\alpha})_i - \nabla f(\boldsymbol{\alpha})_j| \leq 2 \max_{\boldsymbol{z}_k} |\tilde{k}(\boldsymbol{z}_i, \boldsymbol{z}_k) - \tilde{k}(\boldsymbol{z}_j, \boldsymbol{z}_k)| \leq \|\tilde{\varphi}(\boldsymbol{z}_i) - \tilde{\varphi}(\boldsymbol{z}_j)\|^2$.

7 Conclusion

We studied the problem of finding sparse combinations of distributed elements, focusing on minimizing the communication overhead. To this end, we proposed a distributed Frank-Wolfe algorithm with favorable properties. We established a lower-bound to the communication cost for the family of problems we consider, which shows that dFW is optimal in its dependency on the approximation quality ϵ. Experiments confirmed the practical utility of the approach.

Acknowledgments Computation for the work described in this paper was supported by the University of Southern California's Center for High-Performance Computing (`http://hpcc.usc.edu`). This research is partially supported by NSF grants CCF-0953192 and CCF-1101215, AFOSR grant FA9550-09-1-0538, ONR grant N00014-09-1-0751, a Google Research Award, a Microsoft Research Faculty Fellowship, the Intelligence Advanced Research Projects Activity (IARPA) via Department of Defense U.S. Army Research Laboratory (DoD / ARL) contract number W911NF-12-C-0012, a NSF IIS-1065243, an Alfred. P. Sloan Research Fellowship, DARPA award D11AP00278, and an ARO YIP Award (W911NF-12-1-0241). The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoD/ARL, or the U.S. Government.

References

[1] F. L. Lewis. Wireless sensor networks. *Smart environments: technologies, protocols, and applications*, pages 11–46, 2004.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. J., R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[3] A. C.-C. Yao. Some complexity questions related to distributive computing. In *STOC*, pages 209–213, 1979.

[4] H. Abelson. Lower bounds on information transfer in distributed computations. *Journal of the ACM*, 27(2):384–392, 1980.

[5] M.-F. Balcan, A. Blum, S. Fine, and Y. Mansour. Distributed Learning, Communication Complexity and Privacy. In *COLT*, 2012.

[6] H. Daumé III, J. M. Phillips, A. Saha, and S. Venkatasubramanian. Protocols for learning classifiers on distributed data. In *AISTATS*, pages 282–290, 2012.

[7] Ohad Shamir. Fundamental Limits of Online and Distributed Algorithms for Statistical Learning and Estimation. In *NIPS*, 2014.

[8] A. Lazarevic and Z. Obradovic. Boosting Algorithms for Parallel and Distributed Learning. *Distributed and Parallel Databases*, 11(2):203–229, 2002.

[9] P. A. Forero, A. Cano, and G. B. Giannakis. Consensus-Based Distributed Support Vector Machines. *JMLR*, 11:1663–1707, 2010.

[10] S. P. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *FTML*, 3(1):1–122, 2011.

[11] J. C. Duchi, A. Agarwal, and M. J. Wainwright. Dual Averaging for Distributed Optimization: Convergence Analysis and Network Scaling. *IEEE Transactions on Automatic Control*, 57(3):592–606, 2012.

[12] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal Distributed Online Prediction Using Mini-Batches. *JMLR*, 13:165–202, 2012.

[13] M.-F. Balcan, S. Ehrlich, and Y. Liang. Distributed k-means and k-median clustering on general communication topologies. In *NIPS*, 2013.

[14] T. Yang. Trading Computation for Communication: Distributed Stochastic Dual Coordinate Ascent. In *NIPS*, 2013.

[15] R. Tibshirani. Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. B*, 58(1):267–288, 1996.

[16] C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

[17] F. R. Bach, G. R. G. Lanckriet, and M. I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *ICML*, 2004.

[18] C. Shen and H. Li. On the Dual Formulation of Boosting Algorithms. *TPAMI*, 32(12):2216–2231, 2010.

[19] H. Lee, A. Battle, R. Raina, and A. Y. Ng. Efficient sparse coding algorithms. In *NIPS*, pages 801–808, 2006.

[20] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

[21] K. L. Clarkson. Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. *ACM Transactions on Algorithms*, 6(4):1–30, 2010.

[22] M. Jaggi. Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization. In *ICML*, 2013.

[23] S. Shalev-Shwartz, N. Srebro, and T. Zhang. Trading Accuracy for Sparsity in Optimization Problems with Sparsity Constraints. *SIAM Journal on Optimization*, 20(6):2807–2832, 2010.

[24] M. Jaggi. *Sparse Convex Optimization Methods for Machine Learning*. PhD thesis, ETH Zurich, 2011.

[25] S. Lacoste-Julien and M. Jaggi. An Affine Invariant Linear Convergence Analysis for Frank-Wolfe Algorithms. Technical report, arXiv:1312.7864, 2013.

[26] Aurélien Bellet, Yingyu Liang, Alireza Bagheri Garakani, Maria-Florina Balcan, and Fei Sha. A Distributed Frank-Wolfe Algorithm for Communication-Efficient Sparse Learning. Technical report, arXiv:1404.2644, 2014.

[27] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[28] Robert M. Freund and Paul Grigas. New Analysis and Results for the Conditional Gradient Method. Technical report, arXiv:1307.0873, 2013.

[29] S. K. Shevade and S. S. Keerthi. A simple and efficient algorithm for gene selection using sparse logistic regression. *Bioinformatics*, 19(17):2246–2253, 2003.

[30] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *J. R. Stat. Soc. B*, 68:49–67, 2006.

[31] I. W. Tsang, J. T. Kwok, and P.-M. Cheung. Core Vector Machines: Fast SVM Training on Very Large Data Sets. *JMLR*, 6:363–392, 2005.

[32] H. Ouyang and A. G. Gray. Fast Stochastic Frank-Wolfe Algorithms for Nonlinear SVMs. In *SDM*, pages 245–256, 2010.

[33] S. Asharaf, M. N. Murty, and S. K. Shevade. Multiclass core vector machine. In *ICML*, pages 41–48, 2007.

[34] S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher. Block-Coordinate Frank-Wolfe Optimization for Structural SVMs. In *ICML*, 2013.

[35] D. Mosk-Aoyama, T. Roughgarden, and D. Shah. Fully distributed algorithms for convex optimization problems. *SIAM Journal on Optimization*, 20(6):3260–3279, 2010.

[36] J. N. Tsitsiklis and Z.-Q. Luo. Communication complexity of convex optimization. *Journal of Complexity*, 3(3):231–243, 1987.

[37] E. Wei and A. E. Ozdaglar. Distributed Alternating Direction Method of Multipliers. In *CDC*, pages 5445–5450, 2012.