Communication Efficient Coresets for Empirical Loss Minimization

Sashank J. Reddi
Machine Learning Department
Carnegie Mellon University
sjakkamr@cs.cmu.edu

Barnabás Póczos
Machine Learning Department
Carnegie Mellon University
bapoczos@cs.cmu.edu

Alex Smola
Machine Learning Department
Carnegie Mellon University
alex@smola.org

Abstract

In this paper, we study the problem of empirical loss minimization with l_2-regularization in distributed settings with significant communication cost. Stochastic gradient descent (SGD) and its variants are popular techniques for solving these problems in large-scale applications. However, the communication cost of these techniques is usually high, thus leading to considerable performance degradation. We introduce a novel approach to reduce the communication cost while retaining good convergence properties. The key to our approach is the construction of a small summary of the data, called *coreset*, at each iteration and solve an easy optimization problem based on the coreset. We present a general framework for analyzing coreset-based optimization and provide interesting insights into existing algorithms from this perspective. We then propose a new coreset construction and provide its convergence analysis for a wide class of problems that include logistic regression and support vector machines. Preliminary experiments show encouraging results for our algorithm on real-world datasets.

1 INTRODUCTION

Empirical loss minimization is one of the most fundamental principles in supervised learning. The key idea is to minimize the loss on the training data subject to some regularization on the model that is being learned. More formally, given the training data $P = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ from a probability distribution on $\mathcal{X} \times \mathcal{Y}$, we are interested in the following generic optimization problem:

$$\min_w f(w) \equiv \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \ell(x_i, y_i, w) \qquad (1)$$

Throughout this paper we assume that ℓ is convex. Furthermore, we assume that $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}$. Note that the objective function above is strongly convex (a function f is strongly convex with modulus λ if $f(w) - \frac{\lambda}{2}\|w\|^2$ is a convex function). Problems conforming to Equation (1) include popular supervised learning algorithms like support vector machines and regularized logistic regression. For example, when $x_i \in \mathbb{R}^d$, $y_i \in \{-1, 1\}$ and $\ell(x_i, y_i, w) = \log(1 + \exp(-y_i w^\top x_i))$ (logistic loss), the optimization problem in Equation (1) corresponds to regularized logistic regression. The loss function ℓ is not necessarily smooth as in, for example, support vector machines (SVM) where $\ell(x_i, y_i, w) = \max(0, 1 - y_i w^\top x_i)$ (hinge loss).

Several algorithms have been proposed in the literature for solving optimization problems of the aforementioned form. We will briefly review a few key approaches in Section 2; however, the algorithms are either largely synchronous or communication intensive. For example, one of the popular approaches for solving such optimization problems is stochastic subgradient descent. At each iteration of the algorithm, a single training example is chosen at random and used to determine the subgradient of the objective function. While such an approach reduces the computation complexity at each iteration, the communication cost is prohibitively expensive in distributed environment.

In this paper, we study the problem described in Equation (1) in the setting where the data is distributed across nodes and hence, communication is expensive in comparison to the computation time. *The main theme of this paper is to reduce communication cost by constructing and optimizing over a small summary of the training data — which acts as a proxy for the entire data set*. Such a summary of the training points is called a *coreset*. While this methodology has been successfully applied to data clustering problems like k-means and k-median (we refer the reader to [4, 5] for a comprehensive survey), it remains largely unexplored for supervised learning and optimization problems. The goal of this paper is to advance the frontier in this direction. In light of the above, the primary contributions of this paper are as follows:

- We describe a general framework for designing coreset-based algorithms. This also provides insights into existing algorithms from the coresets viewpoint.
- We propose a novel coreset-based algorithm with low communication cost and provable guarantees on the convergence to the optimal solution.
- We demonstrate the efficiency of the proposed algorithm on a few real-world datasets. In particular, we show that the proposed approach reduces the communication cost significantly.

Our paper is structured as follows. We begin with a discussion on the related work in Section 2. In Section 3, we describe a general framework for the coreset-based methodology. We then propose a coreset-based algorithm in Section 4 and provide its convergence analysis. We finally conclude by demonstrating the empirical performance of the algorithm in Section 5.

2 RELATED WORK

As noted earlier, problem in Equation (1) arises frequently in the machine learning and optimization literatures and hence has been a subject of extensive research. Consequently, we cannot hope to do full justice to all the related work. We instead mention the key relevant works here and refer the reader to the appropriate references for a more thorough coverage.

First-order methods: In large-scale machine learning and convex optimization applications, first-order methods are popular due to their cheap iteration cost. The classic approach in first order methods is the gradient descent approach. For strongly convex functions f with L-lipschitz gradient, gradient descent has linear convergence rate i.e., $f(w_t) - f(w_*) \leq \epsilon$ in $O(\log(1/\epsilon))$ iterations where w_t and w_* are the t^{th} iterate of gradient descent and the optimal solution respectively [10]. The constants can be further improved by the means of accelerating techniques [10]. On the other hand, when ℓ is non-smooth, gradient descent methods have sub-linear convergence rates.

While gradient descent methods have appealing convergence properties, they have two major shortcomings: (1) they require evaluation of n gradients at each iteration, typically leading to high computational cost, and (2) the communication costs is also high. A popular modification of this algorithm in large-scale settings is the stochastic gradient descent. While the computational cost per iteration decreases, the linear convergence property is lost. This is due to the variance introduced by stochasticity of the approach. Recently, there has been a surge in interest to address this issue by incremental methods (see [13, 7]). By reducing the variance, these approaches achieve low iteration complexity while retaining the good convergence properties. However, all these approaches still do not address the other major shortcoming — namely, high communication cost.

Active Set & Cutting plane Methods: Our approach is also related to the classic active set and cutting plane methods used in the optimization and the machine learning literatures [11]. The basic idea is to find a *working set* of constraints, i.e., those inequality constraints of the optimization problem that are either fulfilled with equality or are otherwise important to the optimization problem. These methods are particularly popular in the SVM literature. Scheinberg et al. [12] and Joachims et al. [6] provide more details on these approaches. However, these approaches are inherently sequential and not communication friendly. Moreover, it is observed that these approaches are typically outperformed by subgradient methods [14]. While the basic theme of these methods is similar to that of ours insofar that we compute a similar *summary* of the training data at each iteration, the key distinction is the approach and methodology used in constructing the summary. Moreover, our approach is much more general and can be applied to a wide range of loss functions.

Coresets: Our approach is closely related to the paradigm of coresets used in the theory literature [2, 4, 5]. The basic idea of coresets is to extract a small amount of relevant information from the given data and work on this extracted data. Coresets have been proposed on a variety of data clustering problems such as k-means, k-medians, and projective clustering. This approach is particularly important for NP-hard problems like k-means. For example, coresets of size $O(k/\epsilon^4)$ and independent of n (number of the data points) have been proposed for the k-means problem [2, 4]. If k is small, such an approach makes it possible to find optimal solution of k-means simply by an exhaustive search. Furthermore, coresets can seamlessly handle distributed and streaming settings and hence, are suitable to large-scale real-world applications. We refer the reader to the excellent (but outdated) survey on coresets [1] for more details. Recently, a unifying coreset framework has been proposed for data clustering problems [4], which provides a more comprehensive treatment; interested readers may also refer to the references therein. While there has been some progress in borrowing ideas from coresets in the context of SVMs [15], this intersection remains largely unexplored.

Distributed Methods: Owing to large-scale machine learning applications, there has been a recent surge of interest in distributed training of models. The basic idea is to solve subproblems in parallel, followed by averaging at each iteration. For example, [17, 9] propose an algorithm with a trade-off between computation and communication costs. The Alternating Direction Method of Multipliers (ADMM) [3] and its variants are also popular approaches that fall in this category. However, these strategies are either synchronous and communication unfriendly since no communication occurs during the compu-

tation phase. Mini-batch approaches have received considerable attention recently. We refer the reader to [8] and references therein for a more thorough analysis of mini-batch approaches based on stochastic gradient descent.

3 A GENERAL FRAMEWORK

We describe our general methodology in this section. Before delving into the details of the framework, we introduce a few definitions and notations in order to simplify our exposition. We denote the objective function in Equation (1) by $f(w; P)$. Recall that P is the training set. The optimal solution of Equation (1) is denoted by w_* i.e.,

$$w_* = \operatorname*{argmin}_w f(w; P) \equiv \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \ell(x_i, y_i, w).$$

We use R_* to denote $\|w_* - w_0\|$, the distance of optimal solution from the initial point. Next, we define the key ingredient in our approach.

Definition 1. *(Coreset) For given functions f and g, we call a set C an ϵ-coreset of P on a set Ω if $|g(w; C) - f(w; P)| \leq \epsilon$ for all $w \in \Omega$.*

Note that the above definition is slightly different from the one typically used in the coreset literature (see [4]) in two ways: (i) the set C is not necessarily a subset of P. In particular, when the function is of the form described in Equation 1, typically, coresets are constructed for the specific function of g being the weighted sum of loss and coreset C being a subset of P. However, this is not necessarily the case here. (ii) the coreset is restricted to the domain $w \in \Omega$. Another noteworthy point is that while coresets are classically defined as a multiplicative approximation, we use the notion of additive approximation. All these relaxations allow us to view other related algorithms through the lens of coresets. The key desirable property of a coreset is that the cardinality of the set C is small, which will help us reduce the overall communication complexity of the algorithm. For brevity, we drop C and P from the notations $g(w; C)$ and $f(w; P)$ respectively whenever C and P are clear from the context.

With this background we are ready to state our algorithm. At each iteration of the algorithm, the key component of our framework is to compute a new coreset-based on the current solution and solve the optimization problem based on that coreset. The pseudocode is given as Algorithm 1.

First, we note that algorithm is still abstract because it does not specify details about the function $g_{t-1}(w; C_{t-1}, w_{t-1})$ and coreset C_{t-1}. Furthermore, feasible region of the subproblem $\Omega(w_{t-1}, R_{t-1})$ at each iteration is unspecified. These details depend on the specific coreset construction and hence, are explained during the description of the coreset. We now state a general result on performance of Algorithm 1 based on some important properties of the coreset.

Algorithm 1 Generic Iterative Coreset Algorithm

INPUT: Initial w_0, coefficients $\{\gamma_1, \ldots, \gamma_T\}$

1: **for** t = 1 to T **do**
2: Compute the coreset C_{t-1} with the corresponding function $g_{t-1}(w; C_{t-1}, w_{t-1})$
3: Solve the following subproblem

$$w_t = \operatorname*{argmin}_{w \in \Omega(w_{t-1}, R_{t-1})} g_{t-1}(w; C_{t-1}, w_{t-1})$$

4: $R_t = \gamma_t \cdot R_{t-1}$
5: **end for**

For ease of analysis, throughout the paper, we assume that γ_t is chosen in such a way that $R_t = \|w_t - w_*\|$. The analysis for the case where R_t is an upper bound on $\|w_t - w_*\|$ is similar.

Theorem 1. *Suppose we have the following conditions on the function g_{t-1} for $1 \leq t \leq T$:*

1. *g_{t-1} is an upper bound on f and is strongly convex with modulus λ'_{t-1}, for some $\lambda'_{t-1} > 0$.*
2. *The feasible region $\Omega(w_{t-1}, R_{t-1})$ is convex and contains the optimal solution w_*.*
3. *C_{t-1} is an Δ_{t-1}-coreset of P on $\Omega(w_{t-1}, R_{t-1})$ with respect to functions g_{t-1} and f. More precisely, we need $g_{t-1}(w; C_{t-1}, w_{t-1}) \leq f(w; P) + \Delta_{t-1}$ for all $w \in \Omega(w_{t-1}, R_{t-1})$.*

Then for the iterates $\{w_t\}_{t=1}^T$ of Algorithm 1 we have,

$$R_t = \|w_t - w_*\| \leq \sqrt{\frac{2\Delta_{t-1}}{\lambda + \lambda'_{t-1}}}.$$

Proof. We have the following inequalities:

$$g_{t-1}(w_*) \leq f(w_*) + \Delta_{t-1},$$
$$g_{t-1}(w_t) + \langle \partial g_{t-1}(w_t), w_* - w_t \rangle + \frac{\lambda'_{t-1}}{2} \cdot \|w_t - w_*\|^2 \leq g_{t-1}(w_*)$$

The first inequality follows from condition 3 of the theorem. The second inequality follows from the fact that g_{t-1} is strongly convex with modulus λ'_{t-1} (condition 1). Adding the above two inequalities we get

$$g_{t-1}(w_t) + \langle \partial g_{t-1}(w_t), w_* - w_t \rangle + \frac{\lambda'_{t-1}}{2} \cdot \|w_t - w_*\|^2 \leq f(w_*) + \Delta_{t-1}. \quad (2)$$

Because f is strongly convex with modulus λ, we have

$$f(w_*) + \langle \partial f(w_*), w_t - w_* \rangle + \frac{\lambda}{2} \cdot \|w_t - w_*\|^2 \leq f(w_t).$$

Combining it with the fact that g_{t-1} is an upper bound on function f (condition 1), we have

$$f(w_*) + \langle \partial f(w_*), w_t - w_* \rangle + \frac{\lambda}{2} \cdot \|w_t - w_*\|^2 \leq g_{t-1}(w_t). \quad (3)$$

Adding Equations (2) and (3), we get the following.

$$\langle \partial g_{t-1}(w_t), w_* - w_t \rangle + \langle \partial f(w_*), w_t - w_* \rangle + \frac{(\lambda + \lambda'_{t-1})}{2} \cdot \|w_t - w_*\|^2 \leq \Delta_{t-1} \quad (4)$$

To complete the proof we need the following intermediate result.

Lemma 1. *Suppose g_{t-1} satisfies the conditions in Theorem 1, then for iterates w_t, for $1 \leq t \leq T$, of Algorithm 1 we have*

$$\langle \partial g_{t-1}(w_t), w_* - w_t \rangle \geq 0 \quad (5)$$

$$\langle \partial f(w_*), w_t - w_* \rangle \geq 0 \quad (6)$$

Proof. We prove the inequality in Equation (5). The inequality in Equation (6) can be proved in a similar manner. Let $A = \Omega(w_{t-1}, R_{t-1})$ and $\mathbb{I}_A : \mathbb{R}^d \to \mathbb{R}^+$ be the indicator function corresponding to A i.e.,

$$\mathbb{I}_A(w) = \begin{cases} 0 & \text{if } w \in A \\ +\infty & \text{if } w \notin A \end{cases}$$

Recall that the w_t is the optimal solution of the following:

$$w_t = \operatorname*{argmin}_{w \in A} g_{t-1}(w).$$

From the optimality condition of w_t, we have $\partial g_{t-1}(w_t) + \partial \mathbb{I}_A(w_t) = 0$. Therefore, we have

$$\langle \partial g_{t-1}(w_t), w_* - w_t \rangle = \langle -\partial \mathbb{I}_A(w_t), w_* - w_t \rangle \quad (7)$$

Since A is convex (condition 2 of Theorem 1), the subgradient will be the normal cone of A. Using the fact that $w_*, w_t \in A$ (condition 2 of Theorem (1)) and from the definition of the normal cone, we have

$$\langle -\partial \mathbb{I}_A(w_t), w_* - w_t \rangle \geq 0.$$

Using the above inequality in Equation (7), we get the required result. □

Using the inequalities from Lemma 1 in Equation (4) it is easy to see that the result follows. □

The above result gives an upper bound on the distance of the iterate w_t in Algorithm 1 from the optimal solution w_*. Note that the bound depends on Δ_{t-1} which in turn typically depends on the optimality of w_{t-1}. It is easy to see that convergence to the optimal solution is possible as long as $\lim_{t \to \infty} \Delta_t = 0$. It is also worth noting that result does not assume anything on the size of the coreset C_t. However, as we shall see, the communication and computation complexity of the algorithm will critically depend on $|C_t|$ at each iteration.

Before discussing our algorithm based on this framework, we consider a popular instantiation of this framework — gradient descent. For this discussion, we assume that the loss function ℓ is differentiable and has L-lipschitz gradient i.e., $\|\partial \ell(x_i, y_i, w) - \partial \ell(x_i, y_i, w')\| \leq L\|w - w'\|$. This smoothness condition on the gradient gives us the following useful result.

Lemma 2. *[10] For any function $h : \mathbb{R}^d \to \mathbb{R}$ with L-Lipschitz continuous gradient ∂h, we have*

$$h(x) \leq h(y) + \langle \partial h(y), x - y \rangle + \frac{L}{2}\|x - y\|^2, \ \forall x, y \in \mathbb{R}^d.$$

The update for gradient descent is the following:

$$w_{t+1} = w_t - \gamma \partial f(w; P), \quad (8)$$

where γ is the learning rate and is typically set to $1/L$. Such an update can be obtained by minimizing the upper bound on f in Lemma 2. We briefly explain how gradient descent like method fits our framework. We choose the coreset[1] $C_t = \partial f(w_t; P)$ and the function g_t as follows:

$$g_t(w; C_t, w_t) = f(w_t; P) + \langle \partial f(w_t), w - w_t \rangle + \frac{L + \lambda}{2} \cdot \|w - w_t\|^2 \quad (9)$$

First note that the function g_t is an upper bound of f and is strongly convex with modulus $L + \lambda$. This can be obtained from Lemma 2. Hence, g_t satisfies condition 1 of Theorem 1. Next, we set $\Omega(w_t, R_t) = \mathcal{B}(w_t, R_t)$ where $\mathcal{B}(w, R)$ represents a ball of radius R centered around w. Since this is convex and $R_t = \|w_t - w_*\|$, it is easy to see that condition 2 of Theorem 1 holds.

Finally, g_{t-1} is an Δ_{t-1}-coreset with $\Delta_{t-1} \leq LR_{t-1}^2/2$. This can be obtained by a straightforward reasoning based on the Taylor expansion of f and Lemma 2. Thus all the conditions of Theorem 1 hold. Hence, using Theorem 1 we obtain the following corollary.

Corollary 1. *The iterates w_t of gradient descent algorithm like algorithm (minimizing upper bound in Equation 9 subject to the constraint on $w \in \Omega(w_t, R_t)$) satisfy*

$$R_t = \|w_t - w_*\| \leq \sqrt{\frac{2LR_{t-1}^2}{2(L + 2\lambda)}} = R_{t-1}\sqrt{\frac{1}{(1 + \frac{2\lambda}{L})}}.$$

[1] Recall that the coreset could be any summary of the data, and not necessarily one of its subsets.

In general, dropping the constraint that $w \in \Omega$, recovers the gradient descent algorithm. The above corollary reproduces the well-known linear convergence rate for gradient descent [10]. Note the dependence of the convergence rate on the condition number L/λ. While the result does not lead to any new convergence rates, it provides an interesting insight that gradient descent can be viewed as solving an optimization problem on a *coreset* based on the gradients at each iteration. However, it is important to note that the communication cost is still high when the condition number is large since the gradient needs to be communicated at each iteration. Hence, gradient descent is not suitable for settings of our interest — that is, distributed settings where communication is expensive.

A natural question that arises is whether we can construct more interesting coresets than the gradients of the function. We provide an affirmative answer to this question in the next few sections.

4 CORESET ALGORITHM

In this section, we propose a new coreset-based algorithm. Before discussing the details of the coreset contribution, it is worth mentioning two additional assumptions; however, we should emphasize that the first assumption is only for the ease of exposition.

1. The loss function ℓ is of the form $\ell(x_i, y_i, w) = \ell(y_i w^\top x_i)$. Note the slight abuse of notation in the usage of ℓ. In what follows, the quantity $y_i w^\top x_i$ is referred to as *margin*.
2. ℓ is L-lipschitz continuous i.e., $|\ell(y_i w^\top x_i) - \ell(y_i w'^\top x_i)| \leq L|y_i w^\top x_i - y_i w'^\top x_i|$.

Loss functions that satisfy the above properties include popular choices such as logistic loss (used in logistic regression) and hinge loss (used in SVM). The significance of these assumptions will become clear as we proceed. We also need the following definitions for our discussion.

Definition 2. *(Cover) We call a set of points S as ϵ-cover of a set of points Q if for all $q \in Q$ there exists a point $s \in S$ such that $\|s - q\| \leq \epsilon$.*

Let $N(x)$ for a point x in the cover denote the set of points in Q that are closer to x than any other point in the cover S. With slight abuse of notation, let $\epsilon(Q) = [S, \beta]$, where S is an ϵ-cover of Q, and β is the vector of cardinalities of the sets $\{N(x)|x \in S\}$. Note that $\|\beta\|_1 = |Q|$.

The key insight to our coreset construction of the algorithm is that typically at each iteration there exist only a few *important* data points that are critical from the optimization perspective. For example, consider an iterative algorithm for SVMs. Intuitively, at each iteration, the points that are close to the margin are crucial in comparison to those away from it. Furthermore, due to the piecewise linear nature of the hinge loss, the points far away from the margin can be represented by a linear function precisely. Hence, it is possible to obtain a good summary of the data through few points near the margin and a linear function. With this intuition, we now present our coreset construction.

We define the set $P' = \{x'_i\}_{i=1}^n$ where $x'_i = y_i x_i$. Our coreset construction consists of two primary steps:

Step 1: Identify points whose loss can be approximated by a linear function and construct a *single* linear function as a coreset for these points. Generally, these are points where gradient approximation is good. We denote such a function by LINEARAPPROX. The description of this function will depend on ℓ.

Step 2: Construct a cover or equivalent functional approximation for the rest of the points in the set P'. Since ℓ is assumed to be lipschitz, such a cover also provides approximation guarantees on the empirical loss on P'.

It should be emphasized that while we use the concept of cover for simplicity, a similar analysis can be carried out for clustering based algorithms. In fact, as we will see later, all our experiments are based on clustering. At each iteration, we use disjoint sets G_t and E_t to denote the points concerned with these two steps respectively. Note that $G_t \cup E_t = P'$. We use $l_t \in \mathbb{R}^d$ to denote the linear approximation of loss for points in G_t. Our coreset is $C_t = (I_t, \beta_t, l_t)$ where $[I_t, \beta_t] = \epsilon_t(E_t)$ and function g_t in Algorithm 1 is as follows.[2]

$$g_t(w; C_t) = \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\left(\sum_{x_e \in I_t} \beta_t^e \ell(w^\top x_e) + w^\top l_t\right) + h_t \tag{10}$$

where $h_t = (2LR_*|E_t|\epsilon + |G_t|\delta + c)/n$ for some $\delta > 0$ and constant c. The pseudocode, based on the above key steps, is given as Algorithm 2. The size of the coreset C_t depends on the cardinality of set E_t.

For simplicity, we assume that $w_0 = 0$ for our analysis. In this case, we have $R_* = \|w_*\|$. One of the key components of Algorithm 2 is the function LINEARAPPROX. As mentioned earlier, in general, this function depends on the loss function ℓ. We choose $\Omega(w_0, R_0) = \mathcal{B}(w_0, R_0)$ and $\Omega(w_t, R_t) = \Omega(w_{t-1}, R_{t-1}) \cap \mathcal{B}(w_t, R_t)$ for $t \in [1, \ldots, T-1]$. Recall that we assume the coefficients $\{\gamma_1, \ldots, \gamma_T\}$ are chosen in a way such that $R_t = \|w_t - w_*\|$. We prove the following result for Algorithm 2. The proof of Theorem 2 relies on result of the generic coreset algorithm, Theorem 1.

Theorem 2. *Suppose g_t is as defined in Equation* (10) *and*

[2]Hereinafter we include the parameter w_t in the coreset description C_t.

Algorithm 2 Iterative Coreset Algorithm

INPUT: Initial w_0, coefficients $\{\gamma_1, \ldots, \gamma_T\}$ and $\{\epsilon_1, \ldots, \epsilon_T\}$

for t = 1 to T **do**

$[G_{t-1}, l_{t-1}] = \textsc{LinearApprox}(P', w_{t-1}, R_{t-1})$

$[I_{t-1}, \beta_{t-1}] = \epsilon_{t-1}(P' \backslash G_{t-1})$

Coreset $C_{t-1} = (I_{t-1}, \beta_{t-1}, l_{t-1})$

Solve the following subproblem

$$w_t = \operatorname*{argmin}_{w \in \Omega(w_{t-1}, R_{t-1})} g_{t-1}(w; C_{t-1})$$

$R_t = \gamma_t \cdot R_{t-1}$

end for

LINEARAPPROX *satisfies the following condition:*

$$\max_{w \in \mathcal{B}(w_t, R_t)} \left| \sum_{x_g \in G_t} \ell(w^\top x_g) - [w^\top l_t + c] \right| \leq |G_t|\delta \quad (11)$$

where $[G_t, l_t] = \textsc{LinearApprox}(P', w_t)$ *and* $c \in \mathbb{R}^d$ *and for all* $t \in \{0, \ldots, T-1\}$*, then we have*

$$R_{t+1} = \|w_{t+1} - w_*\| \leq \sqrt{\frac{2LR_*|E_t|\epsilon_t + |G_t|\delta}{\lambda n}}.$$

Proof. We first observe that g_t (in Equation (10)) is strongly convex with modulus λ. Moreover, g_t is an upper bound on f due to the following relation:

$$\begin{aligned}
&\frac{1}{n}\left(\sum_{x_e \in I_t} \beta_t^e \ell(w^\top x_e) + w^\top l_t\right) + h_t \\
&= \frac{1}{n}\left(\sum_{x_e \in I_t} \beta_t^e \ell(w^\top x_e) + w^\top l_t + 2LR_*|E_t|\epsilon_t + |G_t|\delta + c\right) \\
&\geq \frac{1}{n}\left(\sum_{x_e \in I_t} \beta_t^e \ell(w^\top x_e) + 2LR_*|E_t|\epsilon_t + \sum_{x_g \in G_t} \ell(w^\top x_g)\right) \\
&\geq \frac{1}{n}\left(\sum_{x_p \in E_t} \ell(w^\top x_p) + \sum_{x_g \in G_t} \ell(w^\top x_g)\right) = \frac{1}{n}\sum_{x \in P'} \ell(w^\top x)
\end{aligned}$$

The first inequality follows from the definition of h_t. The second step follows from the condition on LINEARAPPROX in the theorem statement. The third step follows from the fact that ℓ is L-lipschitz continuous and $\|\beta_t\|_1 = |E_t|$. Combining the above with regularization term proves the fact that g_t is an upper bound on f.

It is easy to see that the feasible region $\mathcal{B}(w_t, R_t)$ is convex and contains the optimal solution w_*. To obtain an upper bound on the function g_t, we observe the following:

$$\begin{aligned}
&\frac{1}{n}\left(\sum_{x_e \in I_t} \beta_t^e \ell(w^\top x_e) + w^\top l_t\right) + h_t \\
&\leq \frac{1}{n}\left(\sum_{x_e \in I_t} \beta_t^e \ell(w^\top x_e) + 2LR_*|E_t|\epsilon_t \right. \\
&\qquad\qquad \left. + \sum_{x_g \in G_t} \ell(w^\top x_g) + 2|G_t|\delta\right) \\
&= \frac{1}{n}\left(\sum_{x \in P'} \ell(w^\top x) + 4LR_*|E_t|\epsilon_t + 2|G_t|\delta\right)
\end{aligned}$$

The first and second inequalities follow from the condition of the theorem statement and the lipschitz continuous nature of the loss function ℓ.

Therefore, C_t with the corresponding function g_t is an Δ_t-coreset where $\Delta_t \leq (4LR_*|E_t|\epsilon_t + 2|G_t|\delta)/n$. The above reasoning shows that the function g_t satisfies all the conditions of Theorem 1. Applying Theorem 1 on the function g_t, we get the required result. □

It can be observed that the conditions $\epsilon_T \to 0$ and $\delta \to 0$ as $T \to \infty$ ensure convergence of the algorithm to the optimal solution. In general, we can guarantee that our solution is arbitrary close to the optimal solution by choosing δ and ϵ_t appropriately. Furthermore, we can ensure linear convergence of our algorithm by decreasing ϵ_t by a constant factor at each iteration.

It is also important to study the coreset size and the design choice of ϵ_t and δ since they determine the communication cost of our algorithm. Let $\delta = 2LR_* \min\{\epsilon_1, \ldots, \epsilon_T\}$. For this value of δ, we observe the following:

1. The size of the coreset depends on the cardinality of G_t. In general, larger the cardinality of G_t, smaller is the size of the coreset. Furthermore, as a general rule of thumb, if ℓ is asymptotically linear i.e., $\lim_{|m| \to \infty} |\ell(m) - (cm + d)| = 0$ for some constants c, d, the performance of our algorithm will depend on the rate of asymptotic linearity.
2. We typically require $\epsilon_{t+1} \leq \epsilon_t$ for all $t \in \{0, \ldots, T-1\}$. With such a choice, if $|G_t|$ does not decrease, size of the coreset may increase. However, observe that R_t decreases. Thus, typically more points satisfy Equation (11), and the cardinality of G_t usually decreases.
3. Suppose a subset of P' satisfies Equation (11) at iteration t then it will always satisfy the condition in future iterations. This is due to the fact that feasible region shrinks at each iteration. Hence, size of the coreset is always non-increasing.

While the above remarks provide informal reasoning for the size of the coreset, it does not provide a formal analysis.

In order to gain a better understanding, we discuss the implementation of this algorithm and provide a more formal analysis in the case of logistic regression and SVMs. To this end, let us first discuss the function LINEARAPPROX for specific cases.

LINEARAPPROX for differentiable loss functions: The linear approximation in the differentiable case can be obtained through the first-order Taylor expansion of the loss function. More formally, we have

$$\ell(w^\top x_k) = \ell(w_t^\top x_k) + \partial\ell(w_t^\top x_k)(w^\top x_k - w_t^\top x_k) + \frac{\partial^2\ell(z)}{2}(w^\top x_k - w_t^\top x_k)^2$$

for some $z = \tilde{w}_t^\top x$ where $\|\tilde{w}_t - w_t\| \leq R_t$ since our feasible region satisfies $\|w - w_t\| \leq R_t$. The key step is to bound the term $\partial^2 l(z)$. This bound will depend on the structure of the loss function. We now derive these bounds for logistic regression. We want the following to ensure that the condition in Theorem 2 with $l_t = \sum_{x_k \in G_t} \partial\ell(w_t^\top x_k)x_k$ and $c = \sum_{x_k \in G_t}[\ell(w_t^\top x_k) - \partial\ell(w_t^\top x_k)w_t^\top x_k]$:

$$\frac{\partial^2 l(z)}{2}(w^\top x_k - w_t^\top x_k)^2 \leq \delta$$

for all $x_k \in G_t$. The above statement is true when

$$\frac{\partial^2 l(z)}{2} R_t^2 \|x_k\|^2 \leq \delta \tag{12}$$

This can be obtained by a straightforward application of the Cauchy-Schwartz inequality. The final step is to derive an upper bound on $\partial^2 l(z)$. For logistic loss we have

$$\partial^2 l(z) = \frac{1}{(1+\exp(-z))(1+\exp(z))}.$$

Without loss of generality, we can assume $z > 0$. Then we have $\partial^2 l(z) \leq 1/(1+\exp(z))$. Using the above inequality it is easy to see that Equation (12) is satisfied if

$$\frac{R_t^2\|x_k\|^2}{2(1+\exp(z))} \leq \delta.$$

We observe that

$$\frac{R_t^2\|x_k\|^2}{2(1+\exp(z))} = \frac{R_t^2\|x_k\|^2}{2(1+\exp(w_t^\top x_k + z - w_t^\top x_k))} \leq \frac{R_t^2\|x_k\|^2}{2(1+\exp(w_t^\top x_k)\exp(-R_t\|x_k\|))}$$

This follows from the fact that $z = \tilde{w}_t^\top x$ where $\|\tilde{w}_t - w_t\| \leq R_t$. Hence, for logistic regression, the goal of LINEARAPPROX is to identify all points satisfying

$$V_t(x_k) = \frac{R_t^2\|x_k\|^2}{2(1+\exp(w_t^\top x_k)\exp(-R_t\|x_k\|)} \leq \delta$$

and place these points in the set G_t. The linear function to be used for approximation is obtained from the first-order Taylor expansion. The pseudocode for LINEARAPPROX in case of logistic regression is given in Algorithm 3.

Algorithm 3 LINEARAPPROX for Logistic Regression

INPUT: P', w_t, R_t

1: $G_t = \{x_k \in P' \mid V_t(x_k) \leq \delta\}$

2: $l_t = -\sum_{x_k \in G_t} \frac{x_k}{(1+\exp(w_t^\top x_k))}$

Furthermore, we can also obtain a relationship between the margin of x_k with respect to the optimal solution and the iteration at which the point x_k moves to the set G_t. We note the following:

$$\begin{aligned}\frac{R_t^2\|x_k\|^2}{2(1+\exp(z))} &= \frac{R_t^2\|x_k\|^2}{2(1+\exp(w_*^\top x_k + z - w_*^\top x_k))} \\ &\leq \frac{R_t^2\|x_k\|^2}{2(1+\exp(M_k^*)\exp(-2R_t\|x_k\|))} \\ &\leq \frac{R_t^2\|x_k\|^2\exp(2R_t\|x_k\|)}{2\exp(M_k^*)} \leq \frac{\exp(3R_t\|x_k\|)}{2\exp(M_k^*)}\end{aligned}$$

where $M_k^* = |w_*^\top x_k|$. The first step follows from triangle inequality and the fact that $z = \tilde{w}_t^\top x$ where $\|\tilde{w}_t - w_t\| \leq R_t$ and $\|\tilde{w}_* - w_t\| \leq R_t$. The final step follows from the fact that $x^2 \leq \exp(x)$ for $x \geq 0$. Therefore, from above inequality it is easy to see that Equation (12) is satisfied when the following holds

$$R_t \leq \max\left\{\frac{\sqrt{\delta}}{\|x_k\|}, \frac{M_k^* + \log(2\delta)}{3\|x_k\|}\right\}. \tag{13}$$

LINEARAPPROX for SVM

For SVM, the implementation of LINEARAPPROX is pretty straightforward. Due to the piecewise linear nature of the hinge loss, the condition in Equation (10) is satisfied with $\delta = 0$ if $w^\top x_k$ is greater than 1 (or less than 1) for the whole feasible region $\|w - w_t\| \leq R_t$. This is satisfied when

$$R_t \leq \frac{|1 - w_t^\top x_k|}{\|x_k\|}$$

The pseudocode for LINEARAPPROX in case of SVM is given in Algorithm 4.

Algorithm 4 LINEARAPPROX for SVM

INPUT: P', w_t, R_t

1: $G_t = \{x_k \in P' \mid R_t \leq |1 - w_t^\top x_k|/\|x_k\|\}$

2: $l_t = -\sum_{x_k \in G_t} \mathbb{I}(w_t^\top x_k < 1)x_k$

Similarly to the case of logistic regression, we analyze how the margin of x_k affects when it get included in the set G_t. For this, note the following:

$$1 - w_t^\top x_k = 1 - w_*^\top x_k - (w_t - w_*)^\top x_k$$

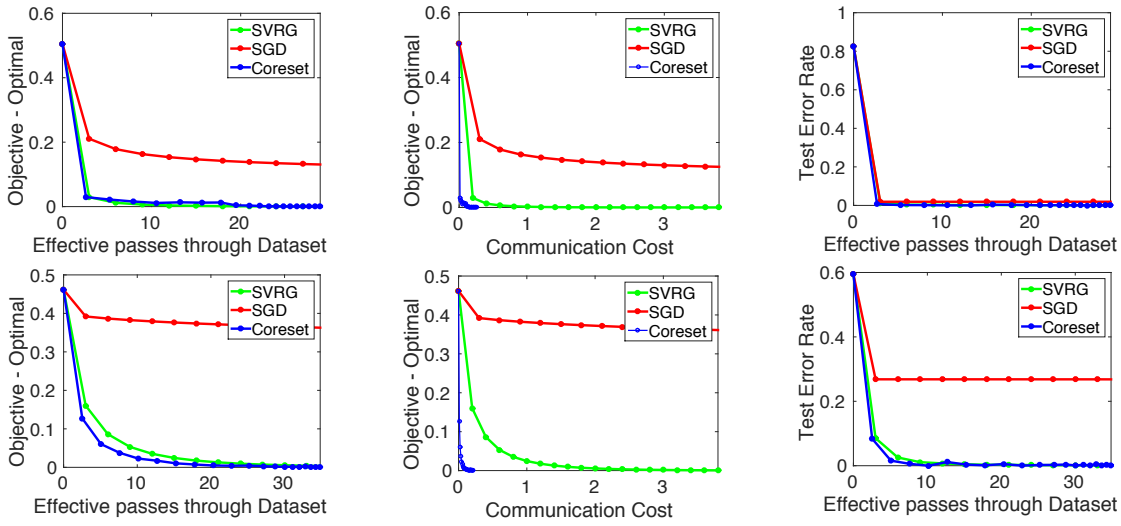

Figure 1: l_2-regularized logistic regression on ijcnn1 (top) and cod-rna (bottom) datasets. We compare our algorithm with mini-batch SVRG and SGD. Training loss residual is shown with respect to passes through the dataset and communication cost (left and central columns). Test error with respect to the passes through the dataset is shown in the right column.

Again, the above quantity will not change sign when $w^\top x_k$ is greater than 1 (or less than 1) for the whole feasible region $\|w - w_t\| \leq R_t$. Based on the expression above, this is satisfied when

$$\|w_t - w_*\|\|x_k\| \leq |1 - w_*^\top x_k| = M_k^*$$

It is obtained by application of the Cauchy-Schwartz inequality. Note the difference in the definition of M_k^* in comparison to logistic regression. Hence, condition in Equation (10) will be satisfied when

$$R_t \leq \frac{M_k^*}{\|x_k\|} \tag{14}$$

Let us make a final remark before proceeding to the experimental section. It should be emphasized that based on Equations (13) and (14), the cardinality of G_t critically depends on the margin of the training points. If the margin of the training points is large, then the coreset size is small and consequently the communication and computation costs are low. Hence, our algorithm is naturally adaptive to the hardness of the optimization problem.

5 EXPERIMENTS

We present our empirical results in this section. To evaluate the performance of our algorithm, we focus on the task of regularized logistic regression. Recall that Equation (1) in this case is of the following form:

$$\min_w f(w) \equiv \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-y_i w^\top x_i)).$$

In our Matlab implementation, we measure simulated communication costs. We use the following datasets.

Dataset	# examples	# features
ijcnn1	49,990	22
cod-rna	59,535	8
w8a	64,700	300
covertype	581,012	54

All these datasets can be accessed from the LIBSVM website.[3] Similarly to [7], each dataset is scaled to $[-1, 1]$. We split each dataset in 3:1 ratio for training and testing purposes respectively.

The regularization parameter λ in Equation (1) is $1/n$. Recall that n is the size of the training set. This results in a high condition number and consequently increases the difficulty of the problem [10]. All the experiments were conducted for 10 random seeds and results are reported by averaging over these 10 runs.

We use PROXSVRG [16] for solving the subproblems of Algorithm 2 at each iteration. SVRG is an incremental first-order method that can be used for solving optimization problems conforming to Equation (1). The origin is used as the initial point for the for all our experiments. The number of "inner iterations" in the PROXSVRG algorithm is set to the recommended value of $m = 2n$. The step size parameter for each dataset is chosen so as to give the fastest convergence for PROXSVRG.

For our experiments, we choose $T = 20$ and $\gamma_1 = \gamma_2 = \cdots = \gamma_T = \gamma$ in Algorithm 2 where γ is chosen such that upper bound on R_T is 0.01. Such a choice is rea-

[3] http://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/

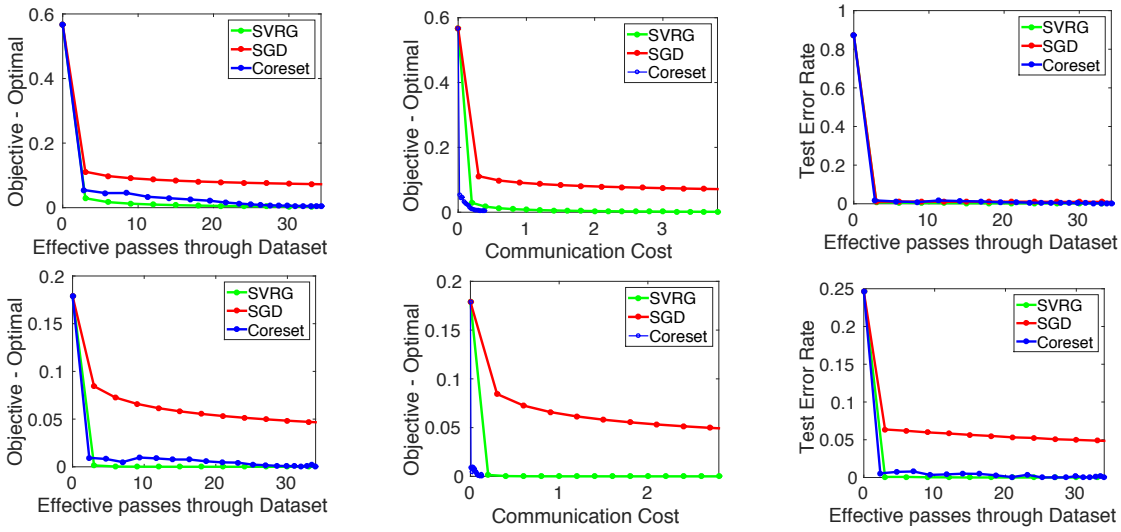

Figure 2: l_2-regularized logistic regression on more datasets w8a (top) and covertype (bottom). Similar to the previous case, we compare our algorithm with mini-batch SVRG and SGD.

sonable in the case of linearly convergent algorithms — which is the scenario we anticipate for our algorithm. As mentioned earlier, we use a clustering based algorithm instead of the cover. The main rationale behind such a choice is the availability of coresets for data clustering problems. As a heuristic, we directly use k-means coresets for Algorithm 2. The sensitivity based coreset for k-means is used in all our experiments. We refer interested reader to [4, 5] for more details of the coreset. We set the coreset size to be 500 for ijcnn1 and cod-rna datasets. This value is set to 1000 and 3000 for w8a and covertype datasets respectively. Note that these coreset sizes are much smaller in comparison to the training data.

We compare our algorithm with SVRG [7] and SGD. A mini-batch version of these methods is used in order to reduce the communication cost of these approaches. We use a mini-batch size $b = 10$ in all our experiments. The number of inner iterations in SVRG is $m = \lceil \frac{2n}{b} \rceil$ in all our experiments in order to limit the total inner iterations to the recommended $2n$ iterations. For SGD, we use the learning rate of $\alpha/\sqrt{t}$ where α is the step size used for the all the algorithms for that dataset.

We report the training loss residual i.e., objective value in Equation (1) achieved by the algorithms minus the optimal objective value (obtained by running gradient descent for a very long time) and the test error rate of the algorithms with respect to the number of effective passes through the dataset . This includes the cost for calculating the gradients and the coresets. This provides information about the computation complexity of the algorithm. To measure the communication cost of the algorithm, we use the ratio of the number of d dimensional vectors communicated to the size of the training data.

Figures 1 and 2 show the performance of the algorithms on the aforementioned datasets. We have several observations from these empirical results. First, we observe that SVRG outperforms SGD in terms of all the metrics of our interest. This observation is not surprising given the linear convergence of SVRG in comparison to the sub-linear convergence of SGD (see [7] for more details). We then observe that our algorithm is competitive to SVRG in terms of training loss residual and test error rate (shown in the first and the third columns of the figures respectively). However, our major gain is in the communication cost of the algorithm. As seen in these figures, our algorithm performs much better in comparison to other algorithms in terms of communication cost. In other words, for the same communication cost, our algorithm has a much lower objective value when compared to SVRG and SGD. We believe that the performance of our algorithm can be further improved by utilizing the coresets of the previous iteration and is a part of our ongoing investigation. For future work, it will be interesting to test the performance of the algorithm on a real distributed environment.

6 CONCLUSION

This paper introduces a novel general strategy for designing communication efficient empirical loss minimization algorithms. The key to our approach is the concept of coresets — the idea of constructing a small summary of the training data and optimizing over this summary. We illustrated this strategy on two popular supervised learning problems — logistic regression and support vector machines. We presented convergence analysis for our algorithm. Furthermore, preliminary experiments show encouraging results in terms of both computational and communication costs.

References

[1] Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Geometric approximation via coresets. In *Combinatorial and Computational Geometry, MSRI*, pages 1–30. University Press, 2005.

[2] Maria-Florina Balcan, Steven Ehrlich, and Yingyu Liang. Distributed k-means and k-median clustering on general communication topologies. In Christopher J. C. Burges, Lon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *NIPS*, pages 1995–2003, 2013.

[3] Stephen Boyd. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2010.

[4] Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 569–578, New York, NY, USA, 2011. ACM.

[5] Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 1434–1453. SIAM, 2013.

[6] Thorsten Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 217–226, New York, NY, USA, 2006. ACM.

[7] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 315–323. Curran Associates, Inc., 2013.

[8] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J. Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 661–670, New York, NY, USA, 2014. ACM.

[9] Dhruv Mahajan, S. Sathiya Keerthi, S. Sundararajan, and Léon Bottou. A functional approximation based distributed learning algorithm. *CoRR*, abs/1310.8418, 2013.

[10] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Mathematics and its applications. Kluwer Academic Publishers, 2004.

[11] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer series in operations research and financial engineering. Springer, 1999.

[12] Katya Scheinberg. An efficient implementation of an active set method for svms. *Journal of Machine Learning Research*, 7:2237–2257, December 2006.

[13] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. Technical report, 2013.

[14] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 807–814, New York, NY, USA, 2007. ACM.

[15] Ivor W. Tsang, James T. Kwok, Pak ming Cheung, and Nello Cristianini. Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*, 6:363–392, 2005.

[16] Lin Xiao and Tong Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, jan 2014.

[17] Martin A. Zinkevich, Alex Smola, Markus Weimer, and Lihong Li. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems 23*, pages 2595–2603, 2010.