Master's Dissertation on

Communication Efficient Distributed Optimization for Regularized Risk Minimization

Submitted in partial fulfillment of the requirements for the degree of

Masters of Technology

by

Laxman Vemula
Roll No: 123059011

under the guidance of

Prof. J. Saketha Nath

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

July 2014

Dissertation Approval Certificate

Department of Computer Science and Engineering

Indian Institute of Technology, Bombay

The dissertation entitled **"Communication Efficient Distributed Optimization for Regularized Risk Minimization"** , submitted by **Laxman Vemula** (Roll No: **123059011**) is approved for the degree of **Master of Technology** in **Computer Science and Engineering** from **Indian Institute of Technology Bombay**.

Prof. J Saketha Nath
Dept. of CSE, IIT Bombay
Supervisor

Prof. Ganesh Ramakrishnan
Dept. of CSE, IIT Bombay
Chairperson & Internal Examiner

Prof. Parag Chaudhuri
Dept. of CSE, IIT Bombay
Internal Examiner

Place : IIT Bombay, Mumbai
Date: 2nd July, 2015

Declaration

I declare that this written submission represents my ideas in my own words and where others ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misinterpreted or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will cause for disciplinary action by the institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Laxman

Signature

Laxman Vemula

Name

123059011

Roll No.

Date: 02/07/2015

Place: Mumbai

Acknowledgement

I would like to express my sincere gratitude to my guide Prof. J. Saketha Nath, for his continuous guidance and support throughout the project. He played a key role in keeping me on the right track and helped me whenever I was clueless on what to do next. I would like to thank Prof. Soumen Chakrabarti, for giving me access to the cluster to run my experiments. I am thankful to Ms. Uma Savant, for helping me to set-up the cluster for experiments and for being there whenever there is an issue with the cluster. I am thankful to my friends and family and all the people who supported me to complete this project work.

Abstract

Distributed learning is now-a-days a widely used framework for training machine learning algorithms on a very large-scale. Distributed learning suffers high communication delay for high-dimensional datasets because very large parameter vectors are to be shared. In this project, we consider distributed learning for the cases where the cost of communication between machines is very high. We propose a formulation that solves the distributed learning problems approximately with low communication cost. We started by introducing partial consensus in Alternating Direction Method of Multipliers (ADMM) which incurs low communication cost but gives approximate solutions. We improved it further by generalizing it to Primal Block Coordinate Descent (ADMM-BCD). We have evaluated our algorithms on high-dimensional datasets and performed a comparison with other existing methods like CoCoA[1], ADMM-Dual Coordinate Ascent (DCA)[2]. The empirical evaluations show that ADMM-BCD performs well in initial iterations but overall ADMM with DCA as local solver is the best method for optimization on high-dimensional datasets.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Numerical optimization plays a key role in training various Machine Learning algorithms. With increasing volume of the training data, both in terms of no. of samples and no. of features, learning became a challenging task.

Various distributed learning methodologies were employed to tackle this problem efficiently. These methods generally split the training data across multiple machines. A naive way is to solve each of the sub-problems independently and get the average of the parameters. There are methods which solve the original problem exactly by achieving consensus on the parameters across the machines. In these methods, each machine iteratively communicates with other machines until consensus is achieved.

For most of the iterative distributed methods, communication cost per iteration is $O(d)$, where d is the feature dimension of the training data. This cost becomes an overhead when d is very large. This incurs a significant delay in communication where the machines spend more time for communication rather than computation.

In this project, we considered distributed learning through consensus-based optimization. In this setting, a set of machines in a network, collaboratively minimize their corresponding objective while having a consensus on the parameters across the machines. We attempt to devise formulations for consensus-based learning that gives approximate solutions with significantly low communication cost. We started with basic idea of Partial Consensus Optimization and improved it upon by proposing the Primal Block Coordinate Descent method using ADMM. We evaluated our algorithms by performing experiments on high-dimensional real world datasets. Also, we have performed a comparative study on various distributed optimization algorithms to see which performs best.

1.0.1 Organization of Report

The report is organized as follows. In Chapter 2, we discuss the basic concepts relevant to the problems considered in the project. In Chapter 3, we discuss the existing approaches for the distributed consensus learning problem. We present our methods in Chapter 4 and the empirical results are discussed in Chapter 5.

Chapter 2

Background

2.1 Linear Binary Classification

Binary classification is defined as task of separating given set of points into two classes based on various attributes possessed by the points. It is called linear binary classification if the separation is done by using simple hyperplane.

Formally, for a given point $\boldsymbol{x_i}$, we assign a label $y_i \in \{-1, +1\}$, using the hyperplane defined by $\boldsymbol{w}^T\boldsymbol{x} + b = 0$, as follows:

$$y_i = sign(\boldsymbol{w}^T\boldsymbol{x_i} + b)$$

Here the hyperplane $(\boldsymbol{w}, b)$ is called as the classifier or the classification model.

Training

Given a set of points with associated labels (also called training set) as

$$\{(\boldsymbol{x_i}, y_i)|\boldsymbol{x_i} \in \mathbb{R}^n, y_i \in \{-1, +1\}, i = 1, 2, \ldots m\}$$

the process of calculating a classification model $(\boldsymbol{w}, b)$ is called the training. There are various types of classifiers named based on how they are trained. In this project we consider training linear binary classifier through statistical learning process called empirical risk minimization. This process builds a model that minimizes the no. of mis-classified points in the training dataset. This can be achieved by solving a convex optimization problem which is discussed in the next section.

2.2 Regularized Risk Minimization

Regularized risk minimization is a generic formulation used to learn most of the machine learning models. This formulation builds a model that minimizes a given loss function over the training dataset while keeping the model as simple as possible.

Let $\mathbb{D} = \{(\boldsymbol{x_i}, y_i) | \boldsymbol{x_i} \in \mathbb{R}^n, y_i \in \{-1, +1\}, i = 1, 2, \ldots m\}$ be a given set of m labelled instances, each instance having n features. The regularized risk minimization for linear binary classification is defined as follows:

$$\min_{\boldsymbol{w} \in \mathbb{R}^{\boldsymbol{n}}} \frac{1}{2} ||\boldsymbol{w}||^2 + C \sum_{i=1}^{m} l(\boldsymbol{w^T x_i}, y_i) \tag{2.1}$$

where l is the loss function which incurs a penalty when an instance is misclassified. $\boldsymbol{w}$ is the parameter vector that defines the linear model. C is the hyper-parameter that controls the trade-off between the loss and the model complexity.

We get the Support Vector Machine (SVM) formulation if we use hinge loss or square hinge loss. We get the logistic regression formulation if we use logistic loss.

2.3 Distributed Learning

Many technological fields like finance, biology and trading etc, are now-a-days generating data in enormous amounts. It has become essential to do data analysis through statistical processes on very large datasets.

Due to emerging distributed file systems like Hadoop[3] and increasing sizes of training data, distributed learning methods are explored recently. These methods take advantage of the data locality in distributed file system where single dataset is partitioned across many machines.

2.3.1 Formulation

Under the distributed setting, the formulation 2.1 can be redefined as follows:

$$\begin{aligned} \min_{\boldsymbol{w_1}, \boldsymbol{w_2}, \ldots, \boldsymbol{w_k} \in \mathbb{R}^{\boldsymbol{n}}} \quad & \frac{1}{2k} \sum_{i=1}^{k} ||\boldsymbol{w_i}||^2 + C \sum_{i=1}^{k} \sum_{j=1}^{m_i} l(\boldsymbol{w_i^T x_{ij}}, y_{ij}) \\ \text{s.t.} \quad & \boldsymbol{w_i} = \boldsymbol{z}, \forall i = 1, 2, \ldots, k \\ & \boldsymbol{z} \in \mathbb{R}^n \end{aligned} \tag{2.2}$$

Where k is the number of machines in the distributed system and m_i is the number of instances in each machine. $\boldsymbol{w_i}$ corresponds to the model parameters for i^{th} node. The equality constraint on the model parameters across the nodes makes this formulation equivalent to 2.1.

In the next chapter, we discuss various methods used to solve this problem.

Chapter 3

Related Work

In this chapter, we discuss the existing methods from the literature which are most suitable to solve the problem 2.1. Then the key challenges in these methods are highlighted to motivate our approach.

3.1 Existing Methods

Here we focused on the synchronous distributed learning methods, where the optimization is operated by a master node by issuing commands to worker nodes with necessary parameters and collecting the results at each iteration. There exist asynchronous methods, in which all the machines communicate with neighbours asynchronously. We did not consider these methods because they are not suitable for the big data platforms like Hadoop.

3.1.1 One-shot Averaging

One-shot averaging is a naive and straight-forward method that approximately optimizes the original objective. In this method, all the nodes locally solve the sub-problem and the end results are averaged to get the final model parameters. This method requires only one time communication of the parameters.

For the Regularized Risk Minimization problem, we solve the following optimization on each of the nodes locally,

$$\boldsymbol{w_i} = \underset{\boldsymbol{w_i}}{\operatorname{argmin}} \frac{1}{2k}||\boldsymbol{w_i}||^2 + C\sum_{j=1}^{m_i} l(\boldsymbol{w_i^T x_{ij}}, y_{ij}) \quad \forall i = 1, 2, \ldots, k \tag{3.1}$$

and compute the average of the parameters to get the result,

$$\bar{\boldsymbol{w}} = \frac{1}{k}\sum_{i=1}^{k} \boldsymbol{w_i}$$

Though this method gives a good approximation when no. of nodes is small, it tends to perform poorly as the no. of nodes increases. Also when the data distribution is skewed among the nodes, this approach gives bad results. For example, consider a case of two nodes where all the positive class instances are in one node and the negative class instances are in the other node. We can not guarantee that the averaged result in close to the optimal. This limitation makes it necessary to perform more iterations by collecting information about all other nodes.

3.1.2 Map Reduce-based Optimization

Map-Reduce is the functional abstraction provided by distributed file systems like Hadoop. This model is inspired by two key abstractions used in functional programming languages namely, Map and Reduce. In the map-step, we apply some function on all the individual nodes. In reduce-step, the results are collected and combined at the master node.

The map-reduce paradigm is best suited for gradient descent based methods.

3.1.2.1 Gradient Descent

Gradient descent is an iterative algorithm used to minimize convex functions. In 2.1, if the loss function is convex and differentiable, the overall formulation becomes convex and differentiable and we can use gradient descent method to minimize the objective.

To simplify the notation, lets define the cumulative loss corresponding to i^{th} node as follows,

$$f_i(\boldsymbol{w}) = \sum_{j=1}^{m_i} l(\boldsymbol{w^T x_{ij}}, y_{ij}) \tag{3.2}$$

Then the gradient of the above function can be written as,

$$\nabla f_i(\boldsymbol{w}) = \sum_{j=1}^{m_i} l'(\boldsymbol{w^T x_{ij}}, y_{ij})\boldsymbol{x_{ij}}$$

Where l is a differentiable convex loss function (e.g., Square Hinge loss). The gradient of the overall objective becomes,

$$\nabla F(\boldsymbol{w}) = \boldsymbol{w} + C \sum_{i=1}^{k} \nabla f_i(\boldsymbol{w})$$

The gradient vector represents the direction of maximum increase at a particular point. The gradient descent method iteratively updates the parameter vector $\boldsymbol{w}$ in the negative direction of gradient as follows,

$$\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t - \lambda \nabla F(\boldsymbol{w}^t)$$

Where λ is called the step size or learning rate, which is used to specify the length of the step to be taken in negative gradient direction.

In distributed setting, it can be observed that the quantity $\nabla F(\boldsymbol{w})$ decomposes over the nodes in the cluster. Hence, this computation is done in parallel by using one map-reduce iteration. In the map-step, the gradient is computed across all the nodes locally, and the results are collected and added up in the reduce-step at the master node. Here, the optimization happens only in the master node, but computation of the required quantities is done in parallel. Hence, the convergence rate of the algorithm remains same as that of gradient descent method. But, there is a per-iteration communication cost of $O(n)$ for worker nodes and $O(nk)$ for the master node where n is the dimension of the parameter vector and k is the no. of nodes.

3.1.3 All Reduce-based Optimization

All Reduce[4] is similar to the Map Reduce method except that the way the nodes communicate with each other is different. This method imposes a tree structure over the nodes in the cluster with master as the root. The nodes are arranged according to their physical proximity. Now the communication from a node happens only with its parent node and the child nodes.

The intuition here is to reduce the high communication cost at the master i.e., $O(nk)$. While collecting the gradient vectors, each node collects the gradient vectors from it's children and adds them up along with it's local gradient vector and passes the result up to the parent. The master node gets the cumulative gradient which is used to take the update step. The updated parameter vector is broadcasted down the tree for the next iteration.

In this method, the per-iteration communication cost at a node is $O(nd)$, where d is the number of children of the node. This method is particularly suitable for a very large cluster, in which the no. of nodes is in the order of thousands.

Apart from gradient descent, All Reduce can be used for any method which requires accumulation of quantities from all the nodes in the cluster. This method improves the communication cost at the master but it is still in the order of dimension of the parameter vector.

3.1.4 Alternating Direction Method of Multipliers

Alternating direction method of multipliers[5](ADMM) is a distributed optimization method based on the concepts of dual decomposition and method of multipliers. This algorithm solves the problems in multiple variables in which the objective can be split into functions of individual variables with global constraints on the variables. The problem can be defined as follows: Let f and g be two convex functions,

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{z}} \quad & f(\boldsymbol{x}) + g(\boldsymbol{z}) \\ \text{s.t.} \quad & A\boldsymbol{x} + B\boldsymbol{z} = \boldsymbol{c} \end{aligned}$$

Where $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{z} \in \mathbb{R}^m$, $A \in \mathbb{R}^{p \times n}$, $B \in \mathbb{R}^{p \times m}$ and $\boldsymbol{c} \in \mathbb{R}^p$. The augmented Lagrangian function for the above problem is defined as follows:

$$L_\rho(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + g(\boldsymbol{z}) + \boldsymbol{\lambda}^T(A\boldsymbol{x} + B\boldsymbol{z} - \boldsymbol{c}) + \frac{\rho}{2}||A\boldsymbol{x} + B\boldsymbol{z} - \boldsymbol{c}||^2 \quad (3.3)$$

Where ρ is called the *penalty parameter* and $\boldsymbol{\lambda}$ is the dual variable. Using the concepts of dual decomposition and method of multipliers, this function is minimized using the following update rules:

$$\begin{aligned} \boldsymbol{x}^{t+1} &= \operatorname*{argmin}_{\boldsymbol{x}} L_\rho(\boldsymbol{x}, \boldsymbol{z}^t, \boldsymbol{\lambda}^t) \\ \boldsymbol{z}^{t+1} &= \operatorname*{argmin}_{\boldsymbol{x}} L_\rho(\boldsymbol{x}^{t+1}, \boldsymbol{z}, \boldsymbol{\lambda}^t) \\ \boldsymbol{\lambda}^{t+1} &= \boldsymbol{\lambda}^t + \rho(A\boldsymbol{x} + B\boldsymbol{z} - \boldsymbol{c}) \end{aligned} \quad (3.4)$$

Here, the optimization is performed by minimizing the augmented Lagrangian alternatively w.r.t. the two primal variables $\boldsymbol{x}$ and $\boldsymbol{z}$. The dual variable is updated in each iteration after z-variable update. This method can be easily applied to the consensus optimization problem 2.2.

3.1.4.1 Consensus Optimization using ADMM

The consensus optimization problem for regularized risk minimization 2.2 can be restated as follows:

$$\begin{aligned} \min_{\boldsymbol{w_1},\boldsymbol{w_2},\ldots,\boldsymbol{w_k}\in\mathbb{R}^{\boldsymbol{n}}} \quad & \sum_{i=1}^{k} g_i(\boldsymbol{w_i}) \\ \text{s.t.} \quad & \boldsymbol{w_i} = \boldsymbol{z}, \forall i = 1, 2, \ldots, k \\ & \boldsymbol{z} \in \mathbb{R}^n \end{aligned} \tag{3.5}$$

Where $g_i(\boldsymbol{w_i}) = \frac{1}{2k}||\boldsymbol{w_i}||^2 + C\sum_{j=1}^{m_i} l(\boldsymbol{w_i^T x_{ij}}, y_{ij})$, corresponds to the local objective at i^{th} node. Here $\boldsymbol{z}$ is the global variable to ensure equality among the parameter vectors across all the nodes.

In lines of the ADMM formulation stated above, we can define the augmented Lagrangian function for 3.5 as follows. For convenience, we used scaled form of ADMM formulation as discussed in [5].

$$L_\rho(\boldsymbol{w_1, w_2, \ldots, w_k}, \boldsymbol{z}, \boldsymbol{\lambda_1}, \boldsymbol{\lambda_2}, \ldots, \boldsymbol{\lambda_k}) = \sum_{i=1}^{k} g_i(\boldsymbol{w_i}) + \frac{\rho}{2}||\boldsymbol{w_i - z + \lambda_i}||^2 \tag{3.6}$$

Here $\boldsymbol{\lambda_i}$s are the dual variables corresponding to each of the nodes. It is easy to see that this objective is decomposable w.r.t. $\boldsymbol{w_i}$ for a fixed $\boldsymbol{z}$. The update rules for the consensus optimization can be written as,

$$\begin{aligned} \boldsymbol{w_i^{t+1}} &= \underset{\boldsymbol{w_i}}{\operatorname{argmin}}\, g_i(\boldsymbol{w_i}) + \frac{\rho}{2}||\boldsymbol{w_i - z^t + \lambda_i^t}||^2 \\ \boldsymbol{z}^{t+1} &= \frac{1}{k}\sum_{i=1}^{k}(\boldsymbol{w_i^{t+1}} + \boldsymbol{\lambda_i^t}) \\ \boldsymbol{\lambda_i^{t+1}} &= \boldsymbol{\lambda_i^t} + \boldsymbol{w_i^{t+1}} - \boldsymbol{z}^{t+1} \end{aligned} \tag{3.7}$$

These rules can be further simplified as follows:

Let $\bar{\boldsymbol{w}}^t = \frac{1}{k}\sum_{i=1}^{k} \boldsymbol{w_i^t}$ and $\bar{\boldsymbol{\lambda}}^t = \frac{1}{k}\sum_{i=1}^{k} \boldsymbol{\lambda_i^t}$ be the average values of the primal and dual variables respectively over all the nodes at t^{th} iteration. Then, from the above rules, we can write

$$\begin{aligned} \boldsymbol{z}^{t+1} &= \bar{\boldsymbol{w}_i}^{t+1} + \bar{\boldsymbol{\lambda}}^t \\ \bar{\boldsymbol{\lambda}}^{t+1} &= \bar{\boldsymbol{\lambda}}^t + \bar{\boldsymbol{w}}_{\boldsymbol{i}}^{t+1} - \boldsymbol{z}^{t+1} \end{aligned}$$

From these two equations we can conclude that $\bar{\boldsymbol{\lambda}}^{t+1} = 0$, that implies that the dual variables will always have the average value zero. Using this result in the update rules above, we can eliminate the variable $\boldsymbol{z}$. The update rules for the consensus optimization problem are,

$$\begin{aligned} \boldsymbol{w_i^{t+1}} &= \underset{\boldsymbol{w_i}}{\operatorname{argmin}}\, g_i(\boldsymbol{w_i}) + \frac{\rho}{2}||\boldsymbol{w_i} - \bar{\boldsymbol{w}}^t + \boldsymbol{\lambda_i^t}||^2 \\ \boldsymbol{\lambda_i^{t+1}} &= \boldsymbol{\lambda_i^t} + \boldsymbol{w_i^{t+1}} - \bar{\boldsymbol{w}}^{t+1} \end{aligned} \tag{3.8}$$

In each iteration, the $\boldsymbol{w_i}$ minimization is performed in parallel at all the worker nodes. The master collects all the parameter vectors and computes the average $\bar{\boldsymbol{w}}$ and is sent back to the worker nodes. The worker nodes update their respective dual variables before moving on to the next iteration.

The $\boldsymbol{w_i}$ minimization can be solved using any convex minimization algorithm like gradient descent. For the case of SVM, we can use efficient dual methods such as dual coordinate accent methods.

In this algorithm, the per iteration communication cost is $O(n)$ at the worker nodes and $O(nk)$ at the master node. This is equivalent to gradient descent but the no. of iterations required for gradient descent is relatively high when compared to ADMM.

3.1.5 ADMM using Dual Coordinate Ascent

It can be observed from the ADMM updates shown in 3.7 that the local problem to solve $\boldsymbol{w_i^{t+1}}$ is very much similar to Linear SVM formulation. This can be efficiently solved in dual by using Dual Coordinate Ascent (DCA). We use the solver presented in [6], to solve the local problem for the case of Square Hinge loss SVM formulation.

As discussed in 3.1.4, the admm updates for Square Hinge loss SVM formulation on i^{th} machine, are as follows.

$$\begin{aligned} \boldsymbol{w_i^{t+1}} &= \underset{\boldsymbol{w_i}}{\operatorname{argmin}}\, g_i(\boldsymbol{w_i}) + \frac{\rho}{2}||\boldsymbol{w_i} - \bar{\boldsymbol{w}}^t + \boldsymbol{\lambda_i^t}||^2 \\ \boldsymbol{\lambda_i^{t+1}} &= \boldsymbol{\lambda_i^t} + (\boldsymbol{w_i^{t+1}} - \bar{\boldsymbol{w}}^{t+1}) \end{aligned} \tag{3.9}$$

Where $g_i(\boldsymbol{w_i}) = \frac{1}{2k}||\boldsymbol{w_i}||^2 + C\sum_{j=1}^{m_i} \max(1 - y_{ij}\boldsymbol{w_i^T x_{ij}}, 0)^2$, corresponds to the local objective at i^{th} node.

Now the problem to be solved in each iteration at each node is,

$$\underset{\boldsymbol{w_i}}{\operatorname{argmin}} \frac{1}{2k}||\boldsymbol{w_i}||^2 + C\sum_{j=1}^{m_i} \max(1 - y_{ij}\boldsymbol{w_i^T x_{ij}}, 0)^2 + \frac{\rho}{2}||\boldsymbol{w_i} - \bar{\boldsymbol{w}}^t + \boldsymbol{\lambda_i^t}||^2 \tag{3.10}$$

The equivalent dual form for the above objective is,

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & f(\boldsymbol{\alpha}) = \frac{h}{2}\boldsymbol{\alpha}^{\boldsymbol{T}}(\boldsymbol{Q}^{\boldsymbol{T}}\boldsymbol{Q} + \boldsymbol{D})\boldsymbol{\alpha} + \rho h(\bar{\boldsymbol{w}}^{\boldsymbol{t}} - \boldsymbol{\lambda}_{\boldsymbol{i}}^{\boldsymbol{t}})^{\boldsymbol{T}}\boldsymbol{Q}\boldsymbol{\alpha} - \mathbf{1}^{\boldsymbol{T}}\boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \leq \alpha_j \leq \infty, \forall j \end{aligned} \tag{3.11}$$

Where,

$$h = \frac{k}{1 + \rho k}$$

$\boldsymbol{Q}$ is matrix with $y_{ij}\boldsymbol{x_{ij}}$ as columns

$\boldsymbol{D}$ is diagonal matrix with $D_{ii} = 1/2C$

The primal and dual variables are related by,

$$\boldsymbol{w_i} = h\boldsymbol{Q}\boldsymbol{\alpha} + \rho h(\bar{\boldsymbol{w}}^{\boldsymbol{t}} - \boldsymbol{\lambda}_{\boldsymbol{i}}^{\boldsymbol{t}}) \tag{3.12}$$

Now, the gradient of the above objective w.r.t $\boldsymbol{\alpha}$ is,

$$\boldsymbol{\nabla f} = \boldsymbol{Q}^{\boldsymbol{T}}(h\boldsymbol{Q}\boldsymbol{\alpha} + \rho h(\bar{\boldsymbol{w}}^{\boldsymbol{t}} - \boldsymbol{\lambda}_{\boldsymbol{i}}^{\boldsymbol{t}})) + \boldsymbol{D}\boldsymbol{\alpha} - \mathbf{1}$$

Which becomes,

$$\boldsymbol{\nabla f} = \boldsymbol{Q}^{\boldsymbol{T}}\boldsymbol{w_i} + \boldsymbol{D}\boldsymbol{\alpha} - \mathbf{1} \tag{3.13}$$

Gradient w.r.t one coordinate α_j is,

$$\nabla_j f = y_j \boldsymbol{w_i^T x_j} + D_{jj}\alpha_j - 1 \tag{3.14}$$

Using this result, we can efficiently calculate the gradient w.r.t. one coordinate if we have the $\boldsymbol{w_i}$ consistent with the dual variables at any intermediate stage. This can be maintained by updating $\boldsymbol{w_i}$ as soon as any dual variable gets updated using the following relation.

$$\boldsymbol{w}_{\boldsymbol{i}}^{t+1} = \boldsymbol{w}_{\boldsymbol{i}}^{t} + (\alpha_j^{new} - \alpha_j^{old})hy_j\boldsymbol{x_j} \tag{3.15}$$

Using this relationship, we maintain both, the primal and dual variables $(\boldsymbol{w_i}, \boldsymbol{\alpha})$ in memory, consistent with each other and perform coordinate ascent w.r.t α_i chosen at random. The algorithm is presented in Algorithm 1 which is very similar to that given in [6] with few additional terms added due to the penalty term induced by ADMM method.

Algorithm 1 Local Dual Coordinate Descent Solver (LocalDCA)

Input: $\boldsymbol{\alpha}, \bar{\boldsymbol{w}}$ and $\boldsymbol{\lambda}$
Data: Local training data $\mathbb{D} = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^{m}$
Initialize $\boldsymbol{\alpha}^0 = \boldsymbol{\alpha}$ and $\boldsymbol{w}^0 = h\sum_{i=1}^{m} \alpha_i y_i \boldsymbol{x_i} + \rho h(\bar{\boldsymbol{w}} - \boldsymbol{\lambda})$
$t = 0$
while $\boldsymbol{\alpha}^t$ is not optimal **do**
  for $i = 1 \ldots m$ **do**
    $t = t + 1$
    $G_i = y_i \boldsymbol{w^T x_i} + D_{ii}\alpha_i - 1$
    $PG_i = \begin{cases} \min(0, G_i) & \text{if } \alpha_i = 0, \\ G_i & \text{Otherwise.} \end{cases}$
    if $PG_i \neq 0$ **then**
      $\alpha_i^{t+1} = \max(\alpha_i^t - G_i/(h\boldsymbol{x_i}^T\boldsymbol{x_i}), 0)$
      $\boldsymbol{w}^{t+1} = \boldsymbol{w}^t + (\alpha_i^{t+1} - \alpha_i^t) h y_i \boldsymbol{x_i}$
    else
      $\boldsymbol{\alpha}^{t+1} = \boldsymbol{\alpha}^t$
      $\boldsymbol{w}^{t+1} = \boldsymbol{w}^t$
    end if
  end for
end while
$\Delta\boldsymbol{\alpha} = \boldsymbol{\alpha}^t - \boldsymbol{\alpha}^0$
$\Delta\boldsymbol{w} = \boldsymbol{w}^t - \bar{\boldsymbol{w}}$
return $(\Delta\boldsymbol{\alpha}, \Delta\boldsymbol{w})$

3.1.5.1 Hot Start

In the equation 4.19, $\boldsymbol{\alpha}$ is updated while solving the local problem using Dual Coordinate Ascent and $\bar{\boldsymbol{w}}$ and $\boldsymbol{\lambda}$ are updated after getting updates from other nodes.

So, for each iteration, instead of starting the local solver with $\boldsymbol{\alpha}$ set to zero, we can use the primal-dual variables $(\boldsymbol{w}, \boldsymbol{\alpha})$ from previous iteration and use it as a starting point. This helps to converge faster, especially in the later iterations of ADMM as the parameters do not change much.

The ADMM algorithm with Hot Start optimization is shown in Algorithm 2.

3.1.6 CoCoA: Communication Efficient Distributed Dual Coordinate Ascent

This method has been proposed by Jaggi et al. [1]. This is a stochastic

Algorithm 2 ADMM using Dual Coordinate Ascent

Data: Training data $\mathbb{D} = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^{m}$ split across k machines
Initialize $\boldsymbol{\alpha}_i^0 = \mathbf{0}$, $\bar{\boldsymbol{w}}^0 = \mathbf{0}$, $\boldsymbol{w}_i^0 = \mathbf{0}$ and $\boldsymbol{\lambda_i} = \mathbf{0}, \forall i = 1 \dots k$
$t = 0$
while $\bar{\boldsymbol{w}}^t$ is not optimal **do**
  for all $i = 1 \dots k$ **do in parallel**
    $(\Delta\boldsymbol{\alpha}_i, \Delta\boldsymbol{w}_i) = \text{LocalDCA}(\boldsymbol{\alpha}_i^t, \bar{\boldsymbol{w}}^t, \boldsymbol{\lambda}_i^t)$
    $\boldsymbol{\alpha}_i^{t+1} = \boldsymbol{\alpha}_i + \Delta\boldsymbol{\alpha}_i$
    $\boldsymbol{w}_i^{t+1} = \bar{\boldsymbol{w}}^t + \Delta\boldsymbol{w}_i$
  end for
  $\bar{\boldsymbol{w}}^{t+1} = \bar{\boldsymbol{w}}^t + \frac{1}{k}\sum_{i=1}^{k} \Delta\boldsymbol{w}_i$ /* Collect using All-Reduce */
  //Broadcast $\bar{\boldsymbol{w}}^t$ to all nodes
  $\boldsymbol{\lambda}_i^{t+1} = \boldsymbol{\lambda}_i^t + \boldsymbol{w}_i^{t+1} - \bar{\boldsymbol{w}}^{t+1} \quad \forall i = 1 \dots k$ /* Update dual variables in parallel */
  $t = t + 1$
end while
return $\bar{\boldsymbol{w}}^t$

Algorithm 3 CoCoA

Input: $T \geq 1$ and Scaling parameter β_k (Default $\beta_k = 1$)
Data: Training data $\mathbb{D} = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^{m}$ split across k machines
Initialize $\boldsymbol{\alpha}_i^0 = \mathbf{0}$, $\boldsymbol{w}^0 = \mathbf{0} \quad \forall i = 1 \dots k$
for $t = 1 \dots T$ **do**
  for all $i = 1 \dots k$ **do in parallel**
    $(\Delta\boldsymbol{\alpha}_i, \Delta\boldsymbol{w}_i) = \text{LocalDCA}(\boldsymbol{\alpha}_i^t, \boldsymbol{w}^t)$
    $\boldsymbol{\alpha}_i^{t+1} = \boldsymbol{\alpha}_i + \frac{\beta_k}{k}\Delta\boldsymbol{\alpha}_i$
  end for
  $\boldsymbol{w}^{t+1} = \boldsymbol{w}^t + \frac{\beta_k}{k}\sum_{i=1}^{k} \Delta\boldsymbol{w}_i$ /* Collect using All-Reduce */
  $t = t + 1$
end for
return $\boldsymbol{w}^t$

distributed method and uses the Dual Coordinate Ascent method shown in Algorithm 1 as the local solver in each node. The difference between this method and other methods is, the dual variables $\boldsymbol{\alpha}$ maintained in this algorithm correspond to the global objective rather than the local objective. After each iteration, The primal parameter vectors are averaged and updated to each node. The dual variables also updated so as to agree with the primal variables.

This primal-dual relationship is maintained throughout the process. The

authors claim that the method is communication efficient. We have included this method here to evaluate this against our methods.

The Algorithm 3 contains the algorithm used by CoCoA solver.

3.2 Motivation

In all the distributed learning methods, the per-iteration communication cost is directly proportional to the feature dimension of the input dataset. When the feature dimension is very large, the communication and accumulation of the parameter vectors incur significant delay. In this scenario, the distributed system spends more time in communicating while the processors being idle. It is hence desirable to reduce the amount of data shared among the nodes in the cluster to speed up the learning process.

This issue motivated us to consider the distributed learning for the cases where the communication cost is high. Here the intuition is that it might not be necessary to share some of the parameters which do not make much difference to the overall objective. Hence, the key is to identify and share only the most important information that is required to get close to the optimal solution of the original problem.

3.3 Problem Statement

The goal of this project is to devise formulations and algorithms for distributed optimization problems that perform way better than one-shot averaging and give near-accurate solutions, with low communication overhead. In other words,

Objective:

- Speed up the distributed learning process by reducing the per-iteration communication cost.
- Provide the theoretical guarantees with necessary assumptions for the proposed approach.
- Evaluate the proposed algorithms by performing empirical analysis on real world datasets and establish a trade-off between the sub optimality and the improvement in communication cost.

Chapter 4

Proposed Method

4.1 Partial Consensus

Motivated by the observations discussed in the previous chapter, we attempted to solve the distributed learning problem 2.2, with a modification to the global constraints. It is easy to verify that in 2.2 the no. of parameters shared across nodes is equal to the number of constraints. The intuition here is that depending upon the distribution of data among the nodes, some of the parameters may agree with each other even without the constraint. Here we reduce the no. of global constraints and there by reducing the communication cost.

The formulation for distributed learning using partial consensus can be formally defined as follows:

$$\begin{aligned} \min_{\boldsymbol{w_1},\boldsymbol{w_2},\ldots,\boldsymbol{w_k}\in\mathbb{R}^{\boldsymbol{n}}} \quad & \frac{1}{2k}\sum_{i=1}^{k}||\boldsymbol{w_i}||^2 + C\sum_{i=1}^{k}\sum_{j=1}^{m_i} l(\boldsymbol{w_i^T x_{ij}}, y_{ij}) \\ \text{s.t.} \quad & \boldsymbol{A w_i} = \boldsymbol{z}, \forall i = 1, 2, \ldots, k \\ & \boldsymbol{z} \in \mathbb{R}^r \\ & \boldsymbol{A} \in \mathbb{R}^{r\times n}, r \leq n \end{aligned} \tag{4.1}$$

Here instead of imposing equality constraint on all the parameters across the nodes, we impose equality constraint on r linear combinations on subsets of parameters using the matrix $\boldsymbol{A}$. This will ensure that the communication cost is $O(r)$. It is desirable to set $r << n$ to get a significant improvement in communication. If $\boldsymbol{A} = \boldsymbol{I}_{n\times n}$ then this formulation is equivalent to 2.2.

Here it is obvious that this formulation will not solve the original problem exactly unless $r = n$. But with the right choice of the matrix $\boldsymbol{A}$ we can come up with a near-accurate solution.

After solving the problem, the parameters across all the nodes may not be equal. So, the average of final parameter vectors is considered as the result.

4.1.1 Implementation

The formulation 4.1 can be efficiently solved using ADMM as discussed in Section. 3.1.4. The augmented Lagrangian function for the formulation 4.1 can be written as follows:

$$L_\rho(\boldsymbol{w_1}, \boldsymbol{w_2}, \ldots, \boldsymbol{w_k}, \boldsymbol{z}, \boldsymbol{\lambda_1}, \boldsymbol{\lambda_2}, \ldots, \boldsymbol{\lambda_k}) = \sum_{i=1}^{k} g_i(\boldsymbol{w_i}) + \frac{\rho}{2}||\boldsymbol{A w_i} - \boldsymbol{z} + \boldsymbol{\lambda_i}||^2 \tag{4.2}$$

Where $g_i(\boldsymbol{w_i}) = \frac{1}{2k}||\boldsymbol{w_i}||^2 + C\sum_{j=1}^{m_i} l(\boldsymbol{w_i^T x_{ij}}, y_{ij})$ and $\boldsymbol{\lambda_i} \in \mathbb{R}^r$.

Using the similar argument presented in 3.1.4, the following update rules for the distribution optimization are derived.

$$\begin{aligned} \boldsymbol{w_i^{t+1}} &= \underset{\boldsymbol{w_i}}{\operatorname{argmin}}\, g_i(\boldsymbol{w_i}) + \frac{\rho}{2}||\boldsymbol{A w_i} - \boldsymbol{z}^t + \boldsymbol{\lambda_i^t}||^2 \\ \boldsymbol{\lambda_i^{t+1}} &= \boldsymbol{\lambda_i^t} + \boldsymbol{A w_i^{t+1}} - \boldsymbol{z}^{t+1} \end{aligned} \tag{4.3}$$

Where $\boldsymbol{z}^t = \frac{1}{k}\sum_{i=1}^{k} \boldsymbol{A w_i^t}$, is the average of the parameters at t^{th} iteration left multiplied by the matrix $\boldsymbol{A}$. The first rule above is local to each node and can be performed in parallel.

4.1.1.1 Solving the sub-problem

The sub-problem to be solved at each node in each iteration is,

$$\boldsymbol{w_i^{t+1}} = \underset{\boldsymbol{w_i}}{\operatorname{argmin}}\, g_i(\boldsymbol{w_i}) + \frac{\rho}{2}||\boldsymbol{A w_i} - \boldsymbol{z}^t + \boldsymbol{\lambda_i^t}||^2 \tag{4.4}$$

This objective is convex in $\boldsymbol{w_i}$ and can be solved using any convex minimization algorithm. For our implementation, we used gradient descent method.

Similar to the argument presented in Section. 3.1.2.1, the gradient of the above function is,

$$\nabla F(\boldsymbol{w_i^t}) = g_i'(\boldsymbol{w_i}) + \rho \boldsymbol{A}^T(\boldsymbol{A w_i} - \boldsymbol{z}^t + \boldsymbol{\lambda_i^t}) \tag{4.5}$$

we can write the equation for update step for parameter $\boldsymbol{w_i}$ as follows,

$$\boldsymbol{w_i^{t+1}} \leftarrow \boldsymbol{w_i^t} - s\nabla F(\boldsymbol{w_i^t})$$

Where s is the step-size which is chosen using the Barzilai and Borwein rule [7]

4.1.2 Effect of free parameters

In the partial consensus problem defined above, the parameters that do not participate in the equality constraint are called unconstrained parameters. In this problem, after the optimization is performed, these unconstrained parameters take different values in different nodes. So, we take an average to get the final parameter vector. In the following section we discuss the theoretical bounds that we were able to achieve.

4.1.2.1 Analysis

Lets analyse this scenario by assuming k machines having the input dataset split across them. Let $\boldsymbol{R_i}$ be the objective due to i^{th} node.

Let $\boldsymbol{w}^*$ be the minimizer of the global objective and $\boldsymbol{r}^*$ be the optimum objective value i.e.,

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_{i=1}^{k} \boldsymbol{R_i(w)}; \qquad \boldsymbol{r}^* = \sum_{i=1}^{k} \boldsymbol{R_i(w^*)} \tag{4.6}$$

Let $\boldsymbol{w}_i^*$ be the minimizer of the local objective at i^{th} node. Let $\boldsymbol{r}^0$ be the sum of the local optimum objectives.

$$\boldsymbol{w}_i^* = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, \boldsymbol{R_i(w)}; \qquad \boldsymbol{r}^0 = \sum_{i=1}^{k} \boldsymbol{R_i(w_i^*)} \tag{4.7}$$

Let $\bar{\boldsymbol{w}}$ be one-shot averaging solution and $\bar{\boldsymbol{r}}$ be the global objective due to one-shot average parameters.

$$\bar{\boldsymbol{w}} = \frac{1}{k} \sum_{i=1}^{k} \boldsymbol{w}_i^*; \qquad \bar{\boldsymbol{r}} = \sum_{i=1}^{k} \boldsymbol{R_i(\bar{w})} \tag{4.8}$$

Clearly by definition, the following holds,

$$\boldsymbol{r^0} \leq \boldsymbol{r^*} \leq \bar{\boldsymbol{r}} \tag{4.9}$$

Let $\bar{\boldsymbol{w}}^{\boldsymbol{p}}$ be the partial consensus solution and let $\boldsymbol{r^p}$ be the corresponding global objective and $\boldsymbol{r^1} = \sum_{i=1}^{k} \boldsymbol{R_i(w_i^p)}$ is sum of local objectives under partial consensus

Assuming each $\boldsymbol{R_i}$ be α-strongly convex and β-smooth, the difference in the objective due to averaging the parameters can be upper and lower bounded by the following inequalities.

$$\frac{\alpha}{2}\sum_{i=1}^{k}\|\boldsymbol{w_i^*}-\bar{\boldsymbol{w}}\|^2 \leq (\bar{\boldsymbol{r}}-\boldsymbol{r^0}) \leq \frac{\beta}{2}\sum_{i=1}^{k}\|\boldsymbol{w_i^*}-\bar{\boldsymbol{w}}\|^2 \tag{4.10}$$

$$\frac{\alpha}{2}\sum_{i=1}^{k}\|\boldsymbol{w_i^p}-\bar{\boldsymbol{w}}^{\boldsymbol{p}}\|^2 \leq (\boldsymbol{r^p}-\boldsymbol{r^1}) \leq \frac{\beta}{2}\sum_{i=1}^{k}\|\boldsymbol{w_i^p}-\bar{\boldsymbol{w}}^{\boldsymbol{p}}\|^2 \tag{4.11}$$

In order to ensure that $\boldsymbol{r^p} < \bar{\boldsymbol{r}}$, the following condition needs to be satisfied.

$$\boldsymbol{r^1}+\frac{\beta}{2}\sum_{i=1}^{k}\|\boldsymbol{w_i^p}-\bar{\boldsymbol{w}}^{\boldsymbol{p}}\|^2 < \boldsymbol{r^0}+\frac{\alpha}{2}\sum_{i=1}^{k}\|\boldsymbol{w_i^*}-\bar{\boldsymbol{w}}\|^2 \tag{4.12}$$

In 4.12, the right hand side of the inequality is a constant and the left hand side varies. It is difficult to ensure that this inequality holds because in general $\beta \geq \alpha$ and it depends on the variance in the parameters. This motivated us to see if we can fix the free parameters which makes the variance in the parameters zero after the optimization.

4.1.2.2 What if the free parameters are fixed?

Now consider another vector $\boldsymbol{w} = \begin{bmatrix}\boldsymbol{w}_1 & \boldsymbol{w}_2\end{bmatrix}^T$ and we fix $\boldsymbol{w}_2$ to the one-shot parameter average and we optimize the global objective w.r.t $\boldsymbol{w}_1$.

$$\tilde{\boldsymbol{w}} = \underset{[\boldsymbol{w}_1\boldsymbol{w}_2]^T;\boldsymbol{w}_2=\bar{\boldsymbol{w}}_2}{\operatorname{argmin}} \sum_{i=1}^{k}\boldsymbol{R_i}(\boldsymbol{w}) \tag{4.13}$$

Now the global objective calculated at $\tilde{\boldsymbol{w}}$ should be better than the objective due to one-shot average.

$$\sum_{i=1}^{k}\boldsymbol{R_i}(\boldsymbol{w^*}) \leq \sum_{i=1}^{k}\boldsymbol{R_i}(\tilde{\boldsymbol{w}}) \leq \sum_{i=1}^{k}\boldsymbol{R_i}(\bar{\boldsymbol{w}}) \tag{4.14}$$

Hence, if we fix the free parameters, and perform distributed consensus on the other parameters, it is guaranteed that we get a solution which is better than the one-shot average. However, this kind of guarantee cannot be ensured if we do not fix the free parameters. This behaviour is observed in experiments also.

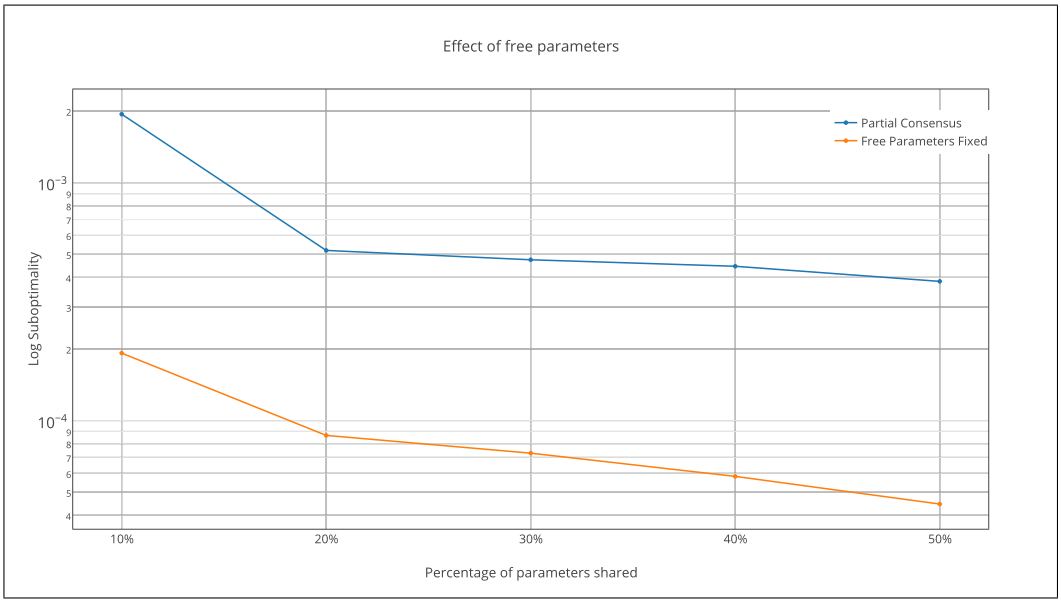


Figure 1: Effect of free parameters

From the empirical evaluations, we realized that the variance in the unconstrained parameters increased after the optimization is performed. Hence, the final average of these unconstrained parameters among the nodes might not be close to the optimal parameters.

The results of this experiment turned out to be better than the partial consensus problem. The results for one dataset are shown in the figure 1. This result motivated us to try Block Coordinate Descent method which is discussed in next section.

4.2 Primal Block Coordinate Descent using ADMM (ADMM-BCD)

Based on the analysis from previous section, we implemented this method. In this method, we select a block of parameters and perform distributed optimization w.r.t that block while keeping the remaining parameters fixed. This is repeated for various choices of the blocks in cyclic fashion. This is analogous to Coordinate Descent but instead of optimizing w.r.t one variable, we select a set of variables. The communication cost in each iteration will be equal to the size of the block chosen.

Let $\boldsymbol{w} \in \mathbb{R}^n$ be the parameter vector and $\boldsymbol{w} = \begin{bmatrix} \boldsymbol{w}^v & \boldsymbol{w}^c \end{bmatrix}^T$ such that $\boldsymbol{w}^v \in \mathbb{R}^r$ and $\boldsymbol{w}^c \in \mathbb{R}^{n-r}$.

Now the optimization problem can be defined as follows:

$$\begin{aligned}
\min_{\boldsymbol{w_1^v},\boldsymbol{w_2^v},\ldots,\boldsymbol{w_k^v}\in\mathbb{R}^{\boldsymbol{n}},\boldsymbol{z}\in\mathbb{R}^{\boldsymbol{r}}} \quad & \sum_{i=1}^{k}(\frac{1}{2k}||\boldsymbol{w_i^v}||^2 + C\sum_{j=1}^{m_i}\xi_{ij}^2) \\
\text{s.t.} \quad & y_{ij}\boldsymbol{w_i}^T\boldsymbol{x_{ij}} \geq 1-\xi_{ij}, \forall i = 1,2,\ldots,k, \forall j = 1,2,\ldots,m_i \\
& \boldsymbol{w_i^v} = \boldsymbol{z}, \forall i = 1,2,\ldots,k \\
& \boldsymbol{w_i^c} = \boldsymbol{b}, \forall i = 1,2,\ldots,k \\
& \boldsymbol{b} \in \mathbb{R}^{n-r}
\end{aligned} \tag{4.15}$$

Where, $\boldsymbol{z}$ is the global variable which ensures equality among the parameters $\boldsymbol{w}_i^v$, across the nodes. The other $n-r$ parameters remain fixed and equal to the vector $\boldsymbol{b}$ in all the nodes.

This optimization can be solved using ADMM and in the similar lines of derivation in 3.1.5. The ADMM update rules are as follows.

$$\begin{aligned}
\boldsymbol{w_i^{v(t+1)}} &= \underset{\boldsymbol{w_i^v}}{\operatorname{argmin}} \frac{1}{2k}||\boldsymbol{w_i^v}||^2 + C\sum_{j=1}^{m_i}\xi_{ij}^2 + \frac{\rho}{2}||\boldsymbol{w_i} - \boldsymbol{z}^t + \boldsymbol{\lambda_i^t}||^2 \\
&\text{s.t.} \quad y_{ij}\boldsymbol{w_i}^T\boldsymbol{x_{ij}} \geq 1-\xi_{ij}, \forall j = 1,2,\ldots,m_i \\
\boldsymbol{z}^{t+1} &= \frac{1}{k}\sum_{i=1}^{k}\boldsymbol{w_i^{v(t+1)}} \\
\boldsymbol{\lambda_i^{t+1}} &= \boldsymbol{\lambda_i^t} + \boldsymbol{w_i^{v(t+1)}} - \boldsymbol{z}^{t+1}
\end{aligned} \tag{4.16}$$

The constraint in the above local problem can be written as,

$$y_{ij}\boldsymbol{w_i^{vT}}\boldsymbol{x_{ij}^v} \geq 1 - c_{ij} - \xi_{ij} \tag{4.17}$$

Where $\boldsymbol{x_{ij}} = \begin{bmatrix}\boldsymbol{x_{ij}^v} & \boldsymbol{x_{ij}^c}\end{bmatrix}^T$ is split according to the indices of $\boldsymbol{w}_i^v$ and $\boldsymbol{w}_i^c$. $c_{ij} = y_{ij}\boldsymbol{w_i^{cT}}\boldsymbol{x_{ij}^c}$ is the constant throughout the optimization and can be precomputed for efficiency.

The equivalent dual form for the above objective is,

$$\begin{aligned}
\min_{\boldsymbol{\alpha}} \quad & f(\boldsymbol{\alpha}) = \frac{h}{2}\boldsymbol{\alpha^T}(\boldsymbol{Q^TQ} + \boldsymbol{D})\boldsymbol{\alpha} + \rho h(\boldsymbol{z^t} - \boldsymbol{\lambda_i^t})^{\boldsymbol{T}}\boldsymbol{Q\alpha} + \boldsymbol{c_i^T\alpha} - \boldsymbol{1^T\alpha} \\
\text{subject to} \quad & 0 \leq \alpha_j \leq \infty, \forall j
\end{aligned} \tag{4.18}$$

Where,

$$h = \frac{k}{1 + \rho k}$$

$\boldsymbol{Q}$ is matrix with $y_{ij}\boldsymbol{x_{ij}^v}$ as columns

$\boldsymbol{D}$ is diagonal matrix with $D_{ii} = 1/2C$

The primal and dual variables are related by,

$$\boldsymbol{w_i^v} = h\boldsymbol{Q\alpha} + \rho h(\boldsymbol{z^t} - \boldsymbol{\lambda_i^t}) \tag{4.19}$$

Now, the gradient of the above objective w.r.t $\boldsymbol{\alpha}$ is,

$$\boldsymbol{\nabla f} = \boldsymbol{Q^T w_i^v} + \boldsymbol{D\alpha} + \boldsymbol{c_i} - \boldsymbol{1} \tag{4.20}$$

Gradient w.r.t one coordinate α_j is,

$$\nabla_j f = y_j \boldsymbol{w_i^{vT} x_j^v} + D_{jj}\alpha_j + c_{ij} - 1 \tag{4.21}$$

This result is similar to that of 3.1.5 and we can use Dual Coordinate Ascent to solve this problem. The Local Solver algorithm for Primal Block Coordinate Descent is given in Algorithm 4. Here it can be observed that the optimization happens w.r.t only r parameters and all other parameters are fixed. The terms involving the fixed parameters are precomputed and can be reused efficiently.

The distributed algorithm for Primal Block Coordinate descent is given in Algorithm 5. Here, we start with the one-shot average parameter vector in each node. At each iteration, we choose a set of parameters and optimize for equality on those parameter while minimizing the objective function.

This process is repeated multiple times by choosing different blocks in cyclic fashion. The key advantage of this method is that ADMM requires less iterations when the no. of constraints is less. Also the communication delay is also relatively less as only r variables are shared in each iteration.

Algorithm 4 Local Dual Coordinate Descent Solver for ADMM-BCD (LocalBCD)

Input: $\boldsymbol{\alpha}, \boldsymbol{z}, \boldsymbol{\lambda}$ and $\boldsymbol{c}$
Data: Local training data $\mathbb{D} = \{(\boldsymbol{x_i^v}, y_i)\}_{i=1}^m$
Initialize $\boldsymbol{\alpha}^0 = \boldsymbol{\alpha}$ and $\boldsymbol{w}^{v0} = h\sum_{i=1}^m \alpha_i y_i \boldsymbol{x_i^v} + \rho h(\boldsymbol{z} - \boldsymbol{\lambda})$
$t = 0$
while $\boldsymbol{\alpha}^t$ is not optimal **do**
  for $i = 1 \ldots m$ **do**
    $t = t + 1$
    $G_i = y_i \boldsymbol{w^{vT}} \boldsymbol{x_i^v} + D_{ii}\alpha_i + c_i - 1$
    $$PG_i = \begin{cases} \min(0, G_i) & \text{if } \alpha_i = 0, \\ G_i & \text{Otherwise.} \end{cases}$$
    if $PG_i \neq 0$ **then**
      $\alpha_i^{t+1} = \max(\alpha_i^t - G_i/(h\boldsymbol{x_i^{vT}}\boldsymbol{x_i^v}), 0)$
      $\boldsymbol{w^{v(t+1)}} = \boldsymbol{w^{v(t)}} + (\alpha_i^{t+1} - \alpha_i^t) h y_i \boldsymbol{x_i^v}$
    else
      $\boldsymbol{\alpha}^{t+1} = \boldsymbol{\alpha}^t$
      $\boldsymbol{w^{v(t+1)}} = \boldsymbol{w^{v(t)}}$
    end if
  end for
end while
$\Delta\boldsymbol{\alpha} = \boldsymbol{\alpha}^t - \boldsymbol{\alpha}^0$
$\Delta\boldsymbol{w^v} = \boldsymbol{w^{v(t)}} - \boldsymbol{z}$
return $(\Delta\boldsymbol{\alpha}, \Delta\boldsymbol{w^v})$

Algorithm 5 ADMM - Primal Block Coordinate Descent

Data: Training data $\mathbb{D} = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^{m}$ split across k machines
Initialize $\boldsymbol{\alpha}_i^0 = \mathbf{0}$, $\bar{\boldsymbol{w}}^0 = \boldsymbol{w}_i^0 = \boldsymbol{w}^{os}, \boldsymbol{z}^t = \mathbf{0}$ and $\boldsymbol{\lambda_i^t} = \mathbf{0}, \forall i = 1 \dots k$
for $p = 1 \dots T$ **do**
  /*Choose block of coordinates from $\boldsymbol{w}$. Let $\boldsymbol{w}^v$ be variable part and $\boldsymbol{w}^c$ be the fixed part */
  $\boldsymbol{w}_i = \begin{bmatrix} \boldsymbol{w}_i^v & \boldsymbol{w}_i^c \end{bmatrix}^T, \forall i = 1 \dots k$
  for all $i = 1 \dots k$ **do in parallel**
    /* Precompute the constant part $\boldsymbol{c}_i$ in parallel */
    $c_{ij} = y_{ij} \boldsymbol{w_i^{cT}} \boldsymbol{x_{ij}^c}, \quad \forall j = 1 \dots m_i$
  end for
  $t = 0$
  while $\boldsymbol{z}^t$ is not optimal **do**
    for all $i = 1 \dots k$ **do in parallel**
      $(\Delta \boldsymbol{\alpha}_i, \Delta \boldsymbol{w}_i^v) = \text{LocalBCD}(\boldsymbol{\alpha}_i^t, \boldsymbol{z}^t, \boldsymbol{\lambda}_i^t, \boldsymbol{c}_i)$
      $\boldsymbol{\alpha}_i^{t+1} = \boldsymbol{\alpha}_i + \Delta \boldsymbol{\alpha}_i$
      $\boldsymbol{w}_i^{\boldsymbol{v}(t+1)} = \boldsymbol{z}^t + \Delta \boldsymbol{w}_i^{\boldsymbol{v}}$
    end for
    $\boldsymbol{z}^{t+1} = \boldsymbol{z}^t + \frac{1}{k} \sum_{i=1}^{k} \Delta \boldsymbol{w}_i^{\boldsymbol{v}}$ /* Collect using All-Reduce */
    //Broadcast $\boldsymbol{z}^{t+1}$ to all nodes
    $\boldsymbol{\lambda}_i^{t+1} = \boldsymbol{\lambda}_i^t + \boldsymbol{w}_i^{\boldsymbol{v}(t+1)} - \boldsymbol{z}^{t+1} \quad \forall i = 1 \dots k$ /* Update dual variables in parallel */
    $t = t + 1$
  end while
end for
return $\boldsymbol{w_i}$

Chapter 5

Experiments

5.1 Framework

We have performed the empirical evaluation on Apache Spark [8] cluster, with input dataset stored in Hadoop file system which exists on the same cluster. The cluster has 23 nodes of which one is used as a master and the remaining 22 nodes are used as slaves. Each node in the cluster has 16GB RAM and 8-core 2.0 GHz processor.

The worker processes are launched by driver program at the master using map-step. By default, Spark doesn't support storing state at the worker node after an iteration, but this is required by our algorithm. Hence, for communicating the parameters and updating the average of parameters, we have used custom implementation of akka actors, a distributed message-passing system.

For a better communication throughput, the nodes are arranged in the form of a tree and All-Reduce is used for aggregating the updates from nodes and broadcasting the average to the workers.

5.2 Datasets

We have used the following datasets collected from the Libsvm repository [9]. The details of the datasets are shown in the following table.

5.3 Methods evaluated

We have evaluated the following methods on the above mentioned datasets by solving L2-Regularized Square Hinge loss SVM.

Name	#Features	#Instances (Train/Test)
url	3,231,961	1,677,291/718,839
kdda	20,216,830	8,407,752/510,302
splice-site	11,725,480	500,000/231,390
news20	1,355,191	11,996/8,000
rcv	47236	677399/20242

Table 5.1: Datasets used for empirical evaluation

5.3.1 ADMM

The ADMM method has been implemented using Dual Coordinate Ascent (DCA) as the local solver for each worker. This solver is adopted from LibLinear package.

5.3.1.1 HotStart

This is an improvement to the traditional ADMM method with some runtime optimizations. Since, the local problem is solved in dual, we can retain both the primal and dual variables from previous iteration and use them as a starting point for current iteration. This speeds up the local solver.

5.3.2 Primal Block Coordinate Descent using ADMM (ADMM-BCD)

This method is an attempt to solve the problem with less communication cost. Here we solve the distributed optimization w.r.t only few selected coordinates while keeping others constant. This results in reduction of the vector length that is shared across nodes in each iteration. This process is repeated for various blocks of coordinates in cyclic fashion. The choice of the coordinate/features is based on the variance observed in the parameters after one-shot training in each node.

5.3.3 CoCoA

This method is proposed by Jaggi et. al.(2014). In this method, the problem is solved using distributed dual coordinate ascent where the dual variables maintained in each node correspond to the global problem.

5.4 Results

5.4.1 Metrics used for comparison

We have used the following three metrics to evaluate the performance of the partial consensus algorithm.

Objective value

Objective value is the most natural choice to evaluate optimization algorithms. Here we comparted the ojective value for various choices of r and for all the four approaches for parameter selection.

Test accuracy

For each of the datasets, we have taken apart a subset of the examples to be used as a test set. We calculated the test accuracy of the model obtained for various choices of r and compared it with the test accuracy obtained by the optimal model.

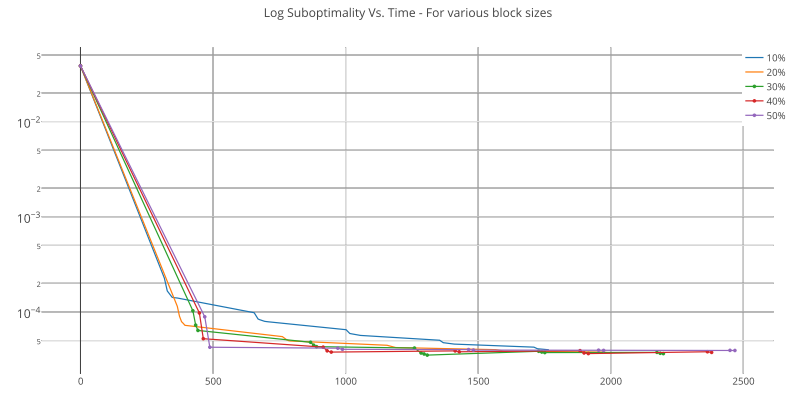
F1-Score

When data is skewed, the test accuracy is not a reliable metric for evaluation of the model. F1-Score gives a better picture on how good the model is.

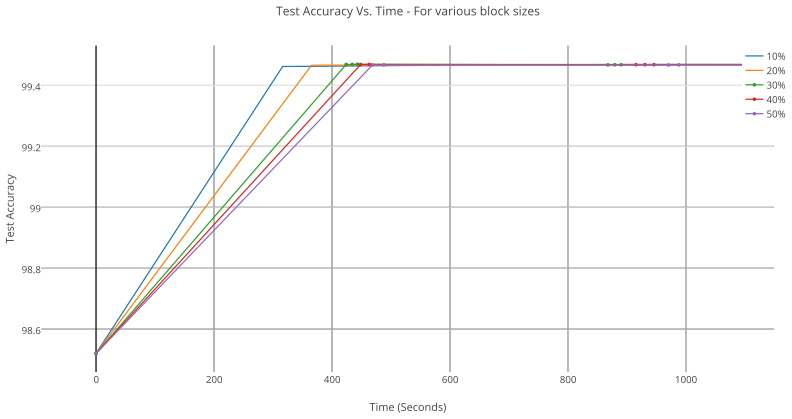
5.4.2 Analysis

5.4.2.1 Effect of the Block Size

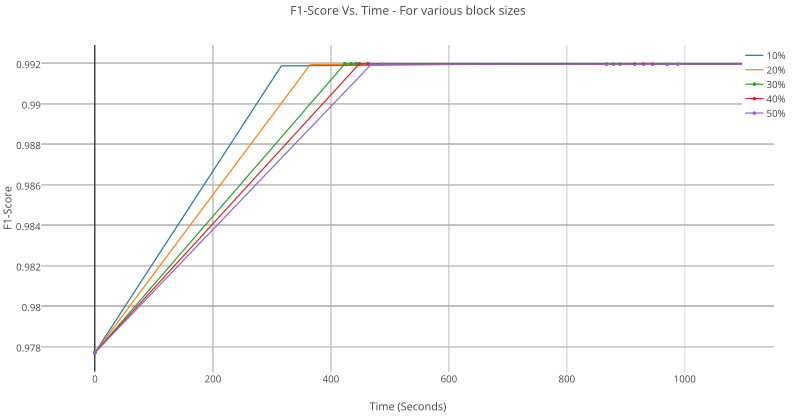
For the URL dataset, we evaluated ADMM-BCD for various block sizes. As shown in the figure, smaller block sizes are better. Another important observation here is that there is significant improvement in the first round and not much later.



(a) Suboptimality



(b) Test Accuracy



(c) F1-Score

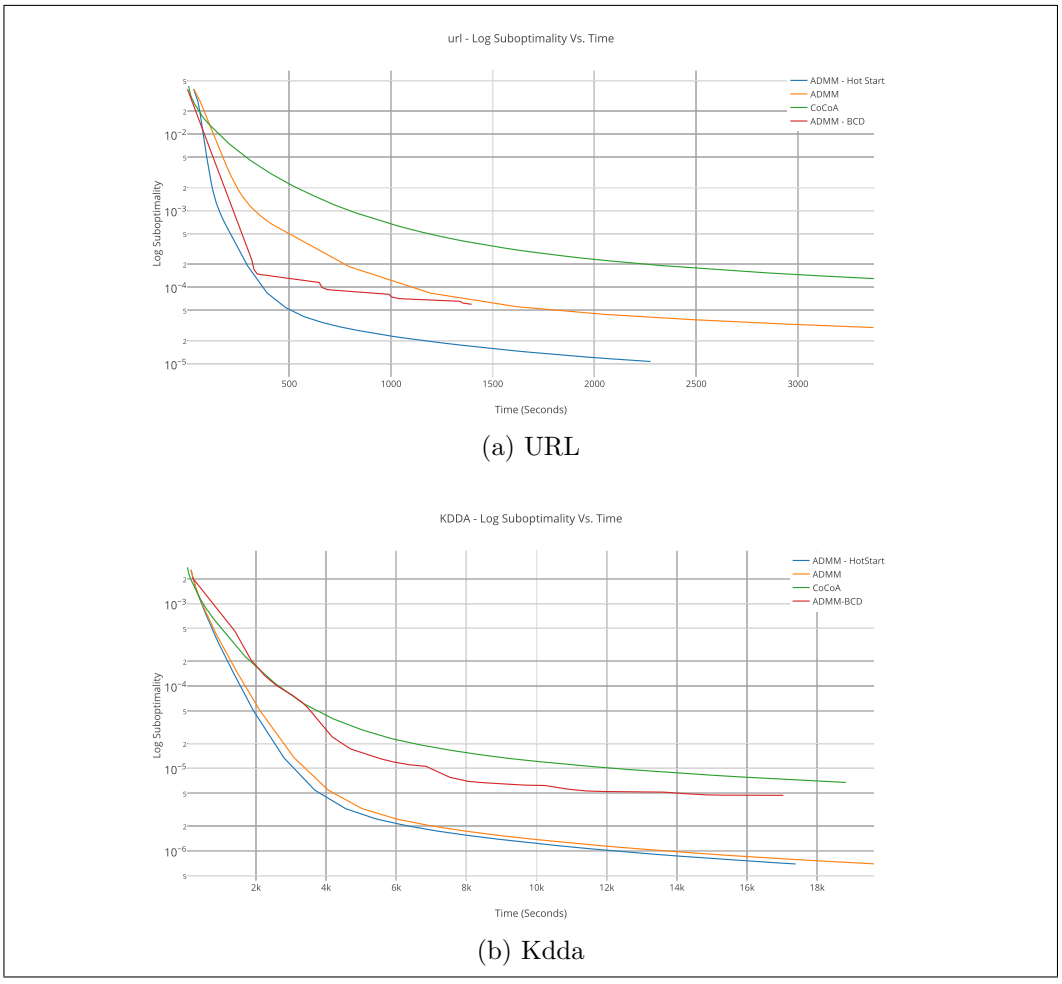
Figure 3: Effect of Block-Size in ADMM-BCD

5.4.2.2 Suboptimality Vs Time

In this section, the suboptimality of the model being trained is presented against the wall clock time. Among the datasets used, three are large datasets namely URL, Kdda and Splice-Site and the other two are relatively small.

From the plots, it is clear that ADMM-HotStart is the best in terms of approaching towards the optimal solution. The effect of HotStart is significant when the number of instances is high. From the plot for News20 dataset, it is clear that there is no difference between ADMM and ADMM-HotStart. This is because News20 has very less no. of instances.

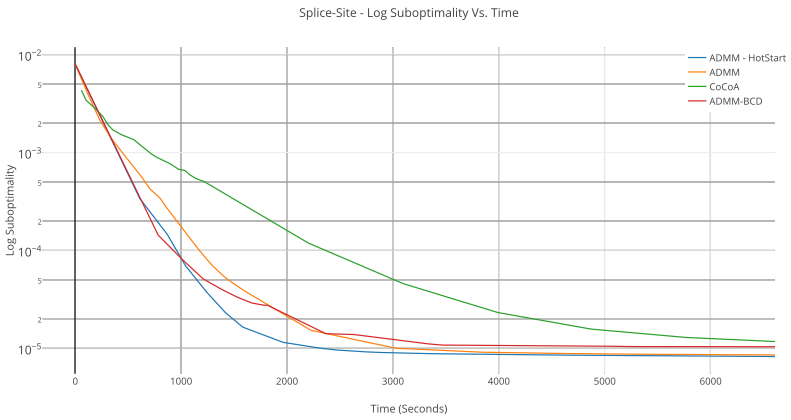
ADMM-BCD performed well on the larger datasets but it is poor when it comes to the smaller datasets. Overall, CoCoA is slower towards convergence than ADMM.



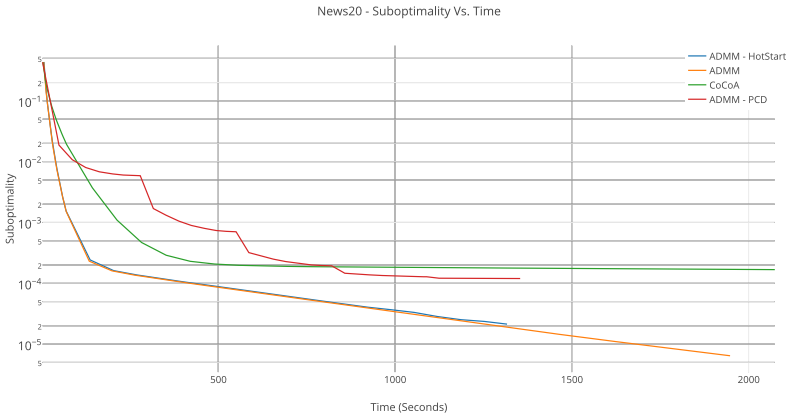
(a) URL

(b) Kdda

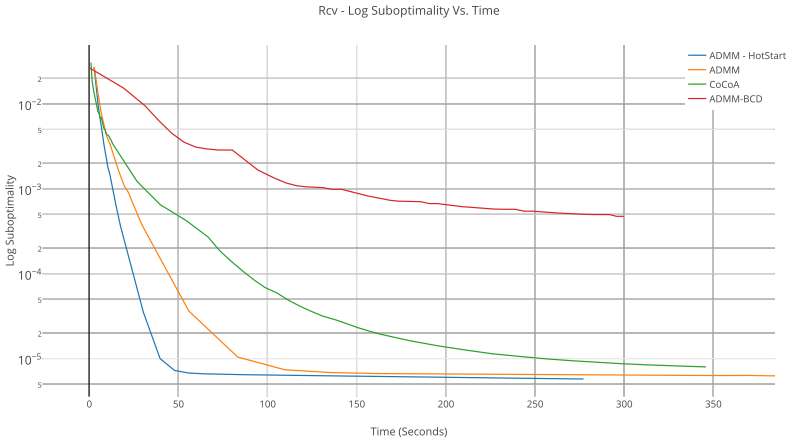
Figure 5: Suboptimality Vs. Time



(c) Splice-Site



(d) News20



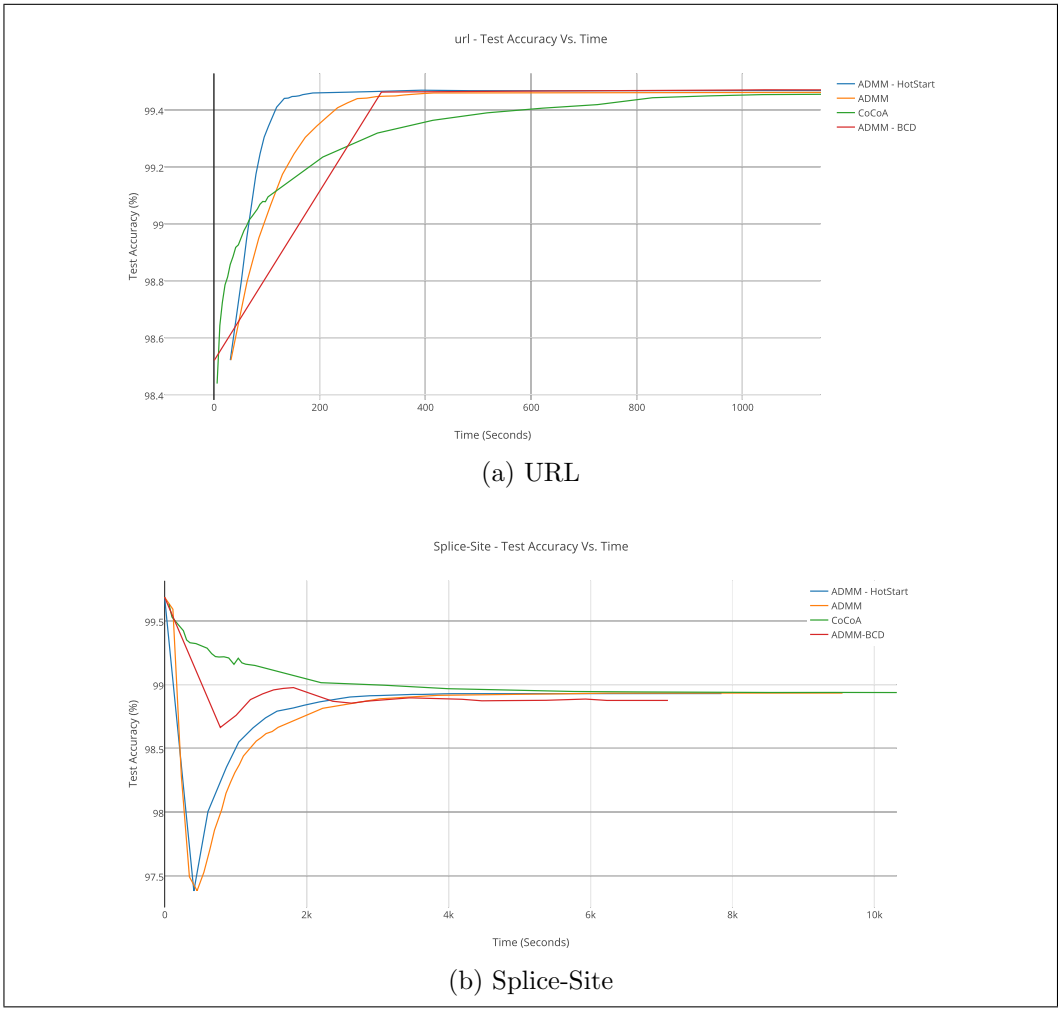
(e) RCV

5.4.2.3 Test Accuracy Vs Time

The test accuracy achieved by each of the algorithms is observed over time, the plots for the four datasets can be found below.

It is easy to see that all the algorithms are improving the test accuracy very quickly in the first few iterations. But, ADMM-HotStart is doing relatively better than others.

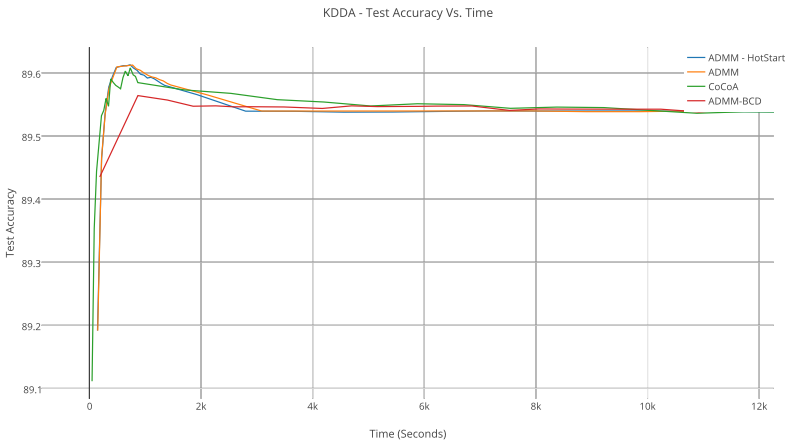
The dataset Splice-Site is a skewed dataset with only 0.1% positive instances. So, it is not a good idea to rely on test accuracy to evaluate the methods for this dataset. F1-Score is a better measure which is presented in next subsection.



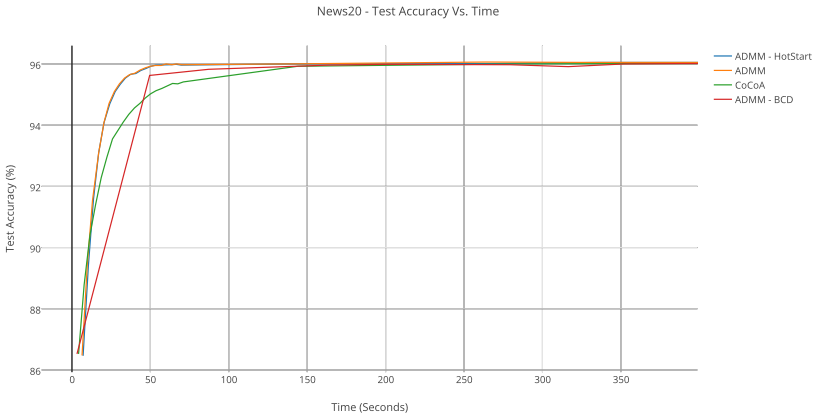
(a) URL

(b) Splice-Site

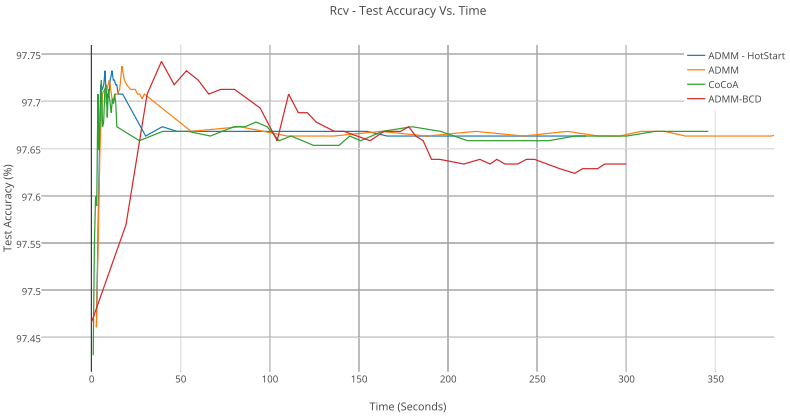
Figure 7: Test Accuracy Vs. Time



(c) Kdda



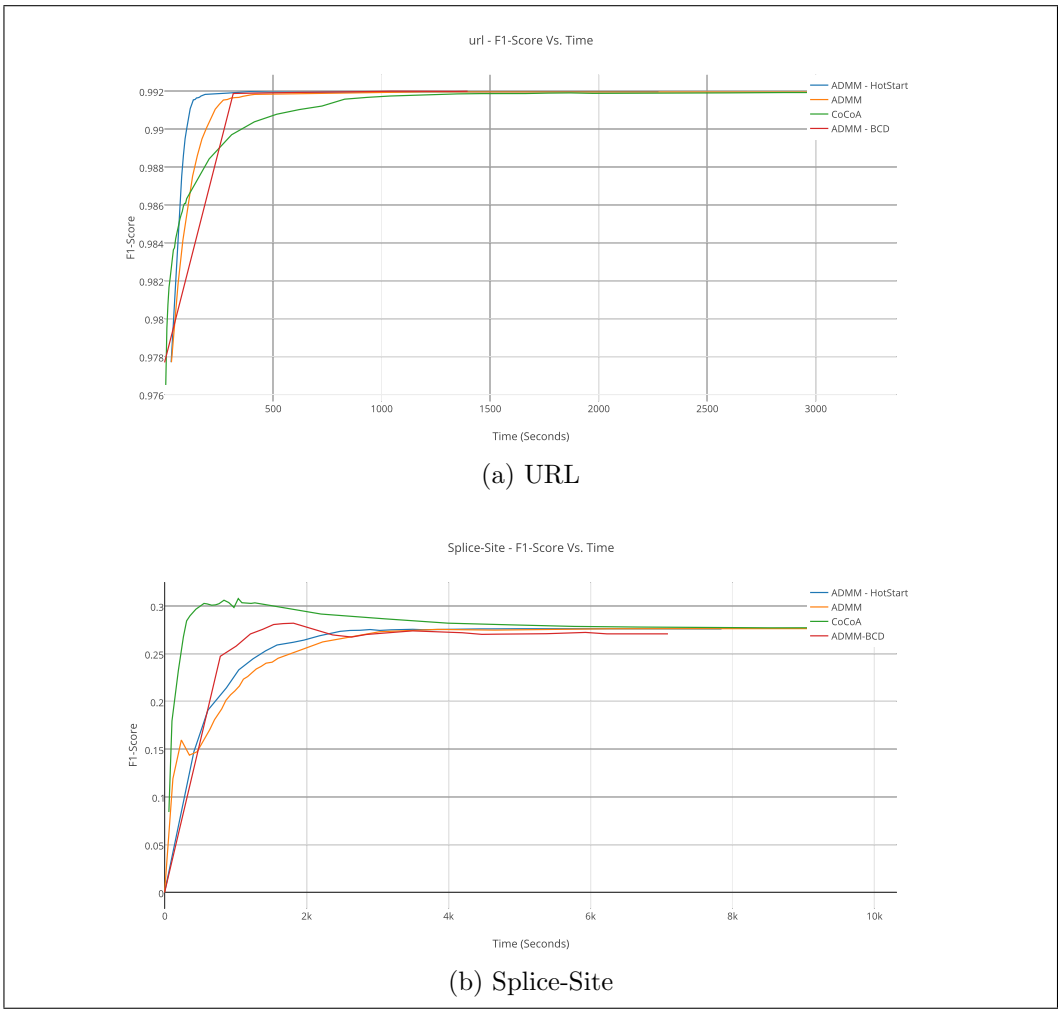
(d) News20



(e) RCV

5.4.2.4 F1-Score Vs Time

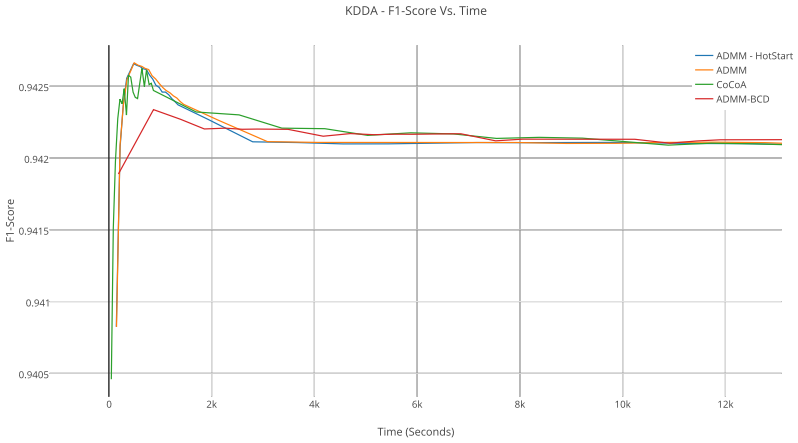
F1-Score is a better measure than test accuracy because it handles imbalanced datasets well. All the methods have shown a very quick improvement in F1-Score. Except for Splice-Site, ADMM reported better F1-Score than other methods. For Splice-Site, CoCoA method reported higher f1-score than other methods during initial iterations, but later it is converged with other methods.



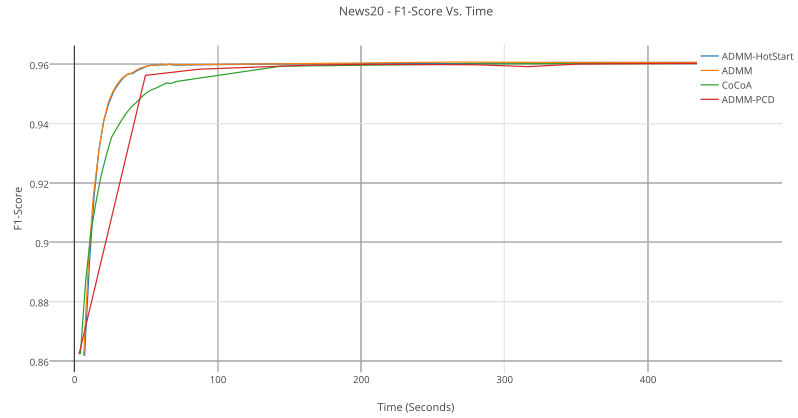
(a) URL

(b) Splice-Site

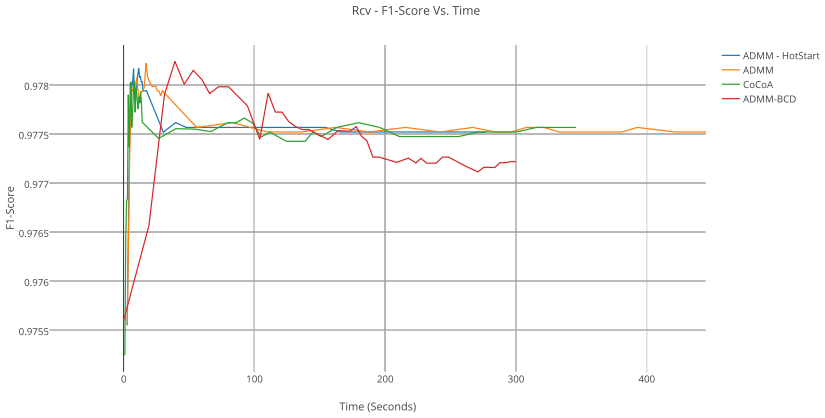
Figure 9: F1-Score Vs. Time



(c) Kdda



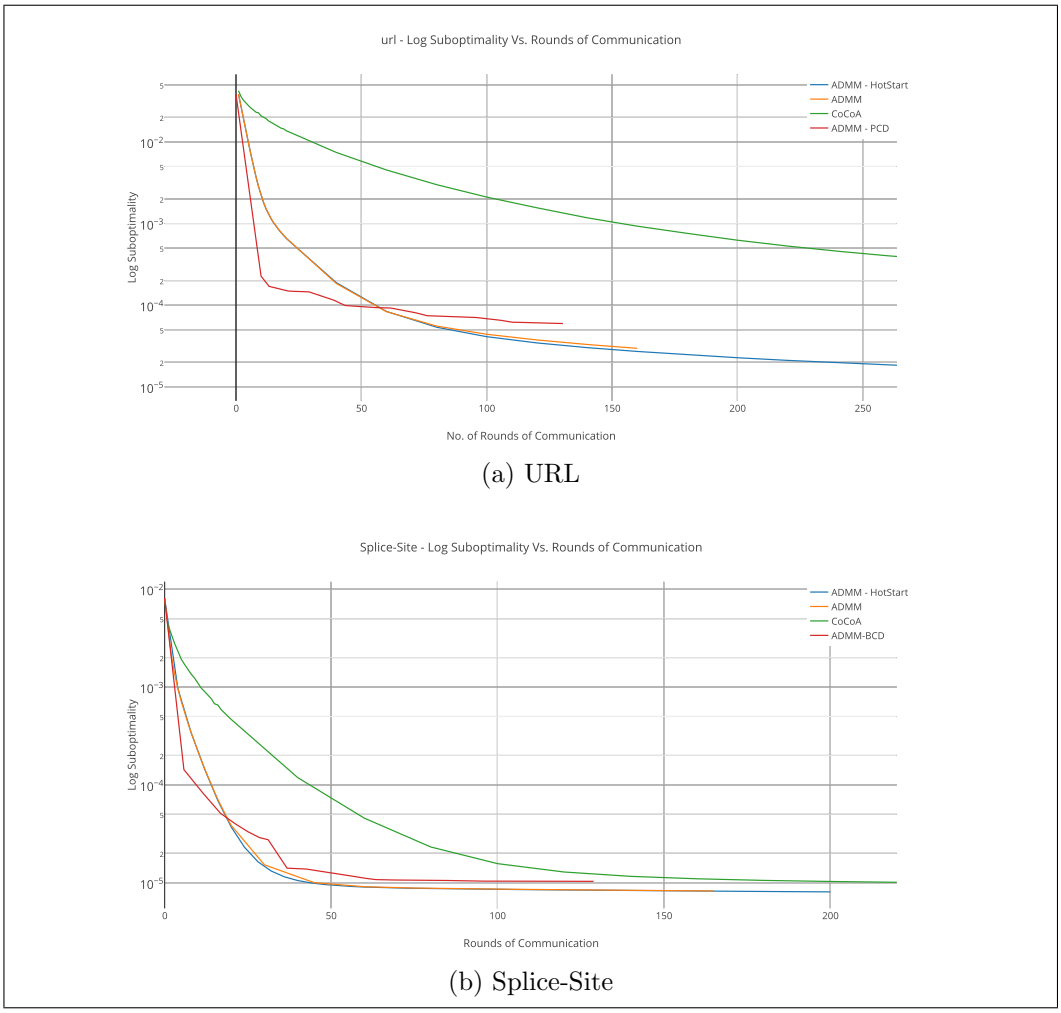
(d) News20



(e) RCV

5.4.2.5 Suboptimality Vs Communication Rounds

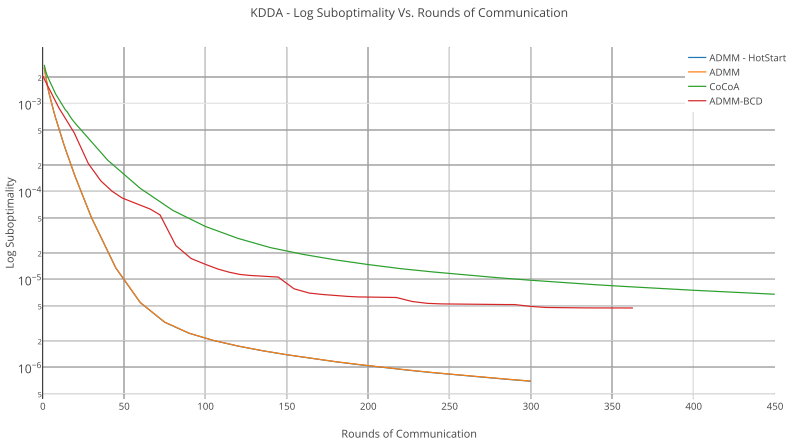
Here we analyse the progress of various algorithms w.r.t amount of data shared across the nodes. It can be observed that ADMM-BCD performs better than all other algorithms in initial few iterations for high-dimensional datasets. But, later it becomes slower than ADMM. However, the optimum test accuracy is attained in the initial few iterations only. In this regard, we can say that ADMM-BCD might work well when the network bandwidth is low.



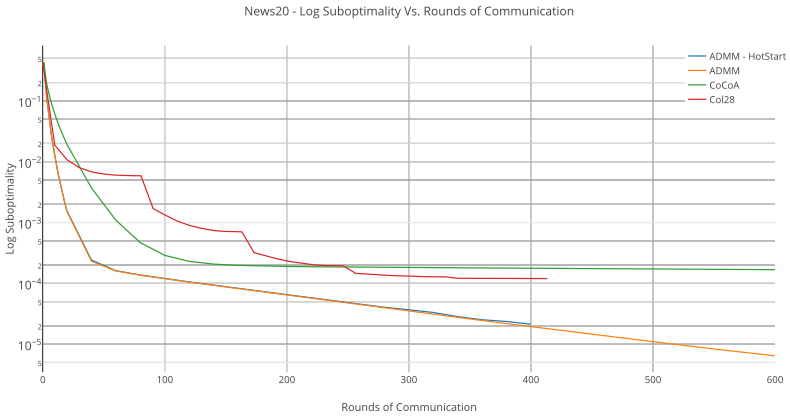
(a) URL

(b) Splice-Site

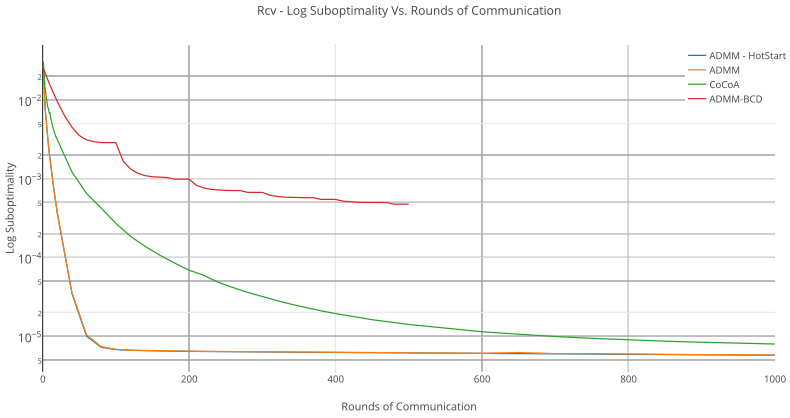
Figure 11: Suboptimality Vs. Communication Rounds



(c) Kdda



(d) News20



(e) RCV

Chapter 6

Conclusion

In this project, we attempted to address the key issue i.e., communication delay, that exists for most of the large-scale distributed optimization algorithms. We started with a formulation called partial consensus optimization which incurs low communication cost and performs better than one-shot averaging. However, we realized that it is better if we fix the free parameters during optimization.

Motivated by this, we tried the method ADMM-BCD. It performs well in the first few iterations but becomes slow later so that the savings in the communication cost become insignificant. However, w.r.t amount of communication, ADMM-BCD performed better than others in initial few iterations for high-dimensional datasets.

From the analysis of empirical results, we can conclude that ADMM with HotStart is the best method. ADMM incurs less communication cost compared to other methods, because the local problem is solved exactly. Whereas, in stochastic methods like CoCoA, the local problem is solved approximately, hence it requires more communications to reach optimum.

Another important observation is that the optimum test accuracy is achieved in very few iterations and didn't improve later. Hence, for binary classification, solving the problem exactly is not required. The distributed optimization can be stopped when we see no improvement in the test accuracy after few iterations.

ADMM has it's drawbacks that it requires an exact solver and also it is slow towards convergence in later iterations. The efficiency of ADMM-HotStart comes from the fact that we use a Dual Coordinate Ascent (DCA) Solver for local problem. DCA is a powerful solver for linear classification and is widely used. Here we use it to improve the distributed optimization for linear classification.

References

[1] M. Jaggi, V. Smith, M. Takác, J. Terhorst, S. Krishnan, T. Hofmann, and M. I. Jordan, "Communication-efficient distributed dual coordinate ascent," *CoRR*, vol. abs/1409.1458, 2014.

[2] C. Zhang, H. Lee, and K. G. Shin, "Efficient distributed linear classification algorithms via the alternating direction method of multipliers," in *International Conference on Artificial Intelligence and Statistics*, pp. 1398–1406, 2012.

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium*, pp. 1–10, May 2010.

[4] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford, "A reliable effective terascale linear learning system," *CoRR*, vol. abs/1110.4198, 2011.

[5] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, pp. 1–122, Jan. 2011.

[6] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan, "A dual coordinate descent method for large-scale linear svm," in *Proceedings of the 25th international conference on Machine learning*, pp. 408–415, ACM, 2008.

[7] J. Barzilai and J. M. Borwein, "Two-point step size gradient methods," *IMA J. Numer. Anal.*, vol. 8, no. 1, pp. 141–148, 1988.

[8] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, (Berkeley, CA, USA), pp. 10–10, USENIX Association, 2010.

[9] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, May 2011.