ADMM based Scalable Machine Learning on Spark

Sauptik Dhar
Research and Technology Center
Robert Bosch LLC
Palo Alto, CA 94304, USA
sauptik.dhar@us.bosch.com

Congrui Yi
Department of Statistics and Actuarial Science
University of Iowa
Iowa City, IA 52242, USA
congrui-yi@uiowa.edu

Naveen Ramakrishnan
Research and Technology Center
Robert Bosch LLC
Palo Alto, CA 94304, USA
naveen.ramakrishnan@us.bosch.com

Mohak Shah
Research and Technology Center
Robert Bosch LLC
Palo Alto, CA 94304, USA
mohak.shah@us.bosch.com

***Abstract*—Most machine learning algorithms involve solving a convex optimization problem. Traditional in-memory convex optimization solvers do not scale well with the increase in data. This paper identifies a generic convex problem for most machine learning algorithms and solves it using the Alternating Direction Method of Multipliers (ADMM). Finally such an ADMM problem transforms to an iterative system of linear equations, which can be easily solved at scale in a distributed fashion. We implement this framework in Apache Spark and compare it with the widely used Machine Learning LIBrary (MLLIB) in Apache Spark 1.3.**

***Keywords*-Distributed Optimization; ADMM; Spark; MLLIB;**

I. Introduction

Convex optimization lies at the core of machine learning algorithms like, linear regression, logistic regression, support vector machines etc. With the advent of big-data, the traditional machine learning algorithms face critical challenges with the continually increasing volume of data. This motivates the need for research in scalable systems and algorithms, particularly suited for solving general classes of convex optimization problems that would in turn help scale machine learning algorithms. There are two aspects of this research which need to be considered jointly as they play a crucial role in the performance of any solution to optimization solvers for the big data setting: 1. Algorithms for distributed optimization, and 2. Systems for big data framework. We briefly describe the state-of-the-art for these aspects and the corresponding choices we make for this paper in the following subsections.

A. Algorithms

Recent years have seen a deluge of novel optimization algorithms for solving big-data machine learning problems. Majority of those approaches follow a distributed framework, and can be broadly categorized as:

1) variations of *stochastic gradient descent* (SGD) [1], [2],
2) *Alternating Direction Method of Multipliers* (ADMM) [3]–[6],
3) approaches that utilize functional approximation based on local portion of the data [7]–[9],
4) Bayesian approaches [10], and
5) Distributed Delayed Optimization [11].

Among all, SGD based approaches have been the most influential and widely used. For example, the Machine Learning Library (MLLIB) packaged with the Spark 1.3 distribution uses SGD [12]. However, current ongoing research and advancements in ADMM presents it as a competitive candidate for such distributed problems [13]–[15]. Unfortunately, very few tools offer any ADMM based distributed machine learning solutions [14], [15]. In this paper we explore the ADMM approach to tackle a generic convex problem, which in turn can be used to solve many machine learning algorithms. We show that, at the heart of the ADMM algorithm is a Quadratic Program (QP) which can be solved in a distributed fashion. The proposed framework provides a scalable solution for a gamut of Machine Learning algorithms (see table I); and is comparable (in terms of computational complexity), to publicly available solutions provided by MLLIB [12].

B. Systems

Another important aspect is the big-data framework used. A variety of architectures have been proposed for Big-Data analytics. On the basis of storage and computation technology, it can be *broadly* categorized as,

1) Single-node in-memory analytics where the entire data is loaded and processed in the memory of a single computer. Such systems require huge amount

S. Dhar and C. Yi contributed equally. This work was done during the course of C. Yi's internship at Robert Bosch LLC, Palo Alto, CA 94304, USA of memory. Typical tools that use such an approach include, MATLAB [16], R [17], KNIME [18], Rapid Miner [19], Weka [20] etc.

2) In-disk analytics where the entire data resides in disk, and chunks of it are loaded and processed in memory. Typical tools that use such an approach are, Revolution R (ScaleR) [21], MATLAB (memmap) [16], GraphLab [22] etc.
3) In-database analytics where the data is stored in a database and the processing is taken to the database where the data resides. Typical tools that use such an approach are, Oracle Data Miner [23], HP Vertica [24], Pivotal [25] etc.
4) Distributed Storage and Computing systems, where the data resides in multiple nodes, and the computation is distributed among those nodes. Typical tools that use such an approach are, Rhadoop (Map-Reduce on Hadoop File System a.k.a HDFS) [26], Mahout (Map-Reduce on HDFS) [27], Apache Spark (distributed in-memory analytics with storage on HDFS) [28], Alpine Data Labs (Map-Reduce/Spark with storage on HDFS) [29] etc.

In this paper we use the Spark computing framework [28] over data stored in HDFS, as it offers several advantages over MapReduce. Specifically, the caching mechanism and lazy execution model of Spark makes it very fast and fault-tolerant, especially for iterative tasks compared to MapReduce which needs to write all intermediate data to the disk. For details on the Spark framework and its performance comparison to MapReduce please refer to [28].

Lately, there has been enormous amount of research on ADMM and it's modifications typically directed towards faster convergence under specific conditions [3], [15]. This paper does not provide new modifications to the ADMM algorithm. The main contribution of this paper includes identifying a generic optimization problem applicable to a gamut of machine learning algorithms (see table I), and using the *standard* ADMM algorithm to solve the optimization in a distributed fashion in Spark. Availability of such a repository of machine learning algorithms in Spark, as an alternative to the currently available Machine Learning LIBrary (MLLIB) [12], can be very useful to the big-data analytics community. We provide the update steps for all the algorithms (in table I), and show that at the core of the ADMM updates is a QP which can be easily solved in a distributed fashion. We benchmark the performance of this generic solver (implemented on Spark 1.3), and compare it with the publicly available Machine Learning LIBrary (MLLIB) for big-data problems.

The rest of the paper is organized as follows. In section II, we introduce the basics of ADMM following [3]. In section III we present the generic optimization problem and provide ADMM updates for a number of machine learning algorithms (shown in Table I). Section IV presents performance comparison of our ADMM implementation and MLlib. Finally we provide the conclusions in Section V.

II. Alternating Direction Method of Multipliers

Alternating Direction Method of Multipliers (ADMM) was first proposed in the mid-70s by Glowinski & Marrocco [6] and Gabay & Mercier [4] as a general convex optimization algorithm. Lately there has been tremendous amount of research in ADMM due to its applicability to the distributed data setting. As an outcome of those research, ADMM presents itself as a competitive technique for distributed optimization. A critical feature of the ADMM formulation is that it divides an optimization problem into smaller sub-problems and enables solutions to them in a distributed setting. Next we present a brief description of the ADMM methodology. A more detailed description can be found in [3].

A. Basic Form

Let's consider optimization problems of the following form [1]:

$$\begin{aligned} \min_{\mathbf{w},\mathbf{z}} \quad & f(\mathbf{w}) + g(\mathbf{z}) \\ \text{s.t.} \quad & A\mathbf{w} + B\mathbf{z} = \mathbf{c} \end{aligned} \tag{1}$$

We form the *augmented Lagrangian* given below,

$$\begin{aligned} L_\rho(\mathbf{w},\mathbf{z},\mathbf{u}) = f(\mathbf{w}) + g(\mathbf{z}) + \mathbf{u}^\top(A\mathbf{w} + B\mathbf{z} - \mathbf{c}) \\ + \frac{\rho}{2}\|A\mathbf{w} + B\mathbf{z} - \mathbf{c}\|_2^2, \end{aligned} \tag{2}$$

where, $\mathbf{u}$ is the lagrange multiplier. Note that, the augmented Lagrangian contains a quadratic penalty term in addition to the usual Lagrangian which is controlled by the penalization factor ρ (see [3] for details). Then the ADMM iterations to solve eq. 1 are,

$$\begin{aligned} \mathbf{w}^{k+1} &= \underset{\mathbf{w}}{\text{argmin}} \quad L_\rho(\mathbf{w},\mathbf{z}^k,\mathbf{u}^k) \\ \mathbf{z}^{k+1} &= \underset{\mathbf{z}}{\text{argmin}} \quad L_\rho(\mathbf{w}^{k+1},\mathbf{z},\mathbf{u}^k) \\ \mathbf{u}^{k+1} &= \mathbf{u}^k + \rho(A\mathbf{w}^{k+1} + B\mathbf{z}^{k+1} - \mathbf{c}) \end{aligned} \tag{3}$$

For practical purposes, a more widely used version is the scaled ADMM. Typically in that case, the linear and the quadratic terms of the primal residual $r = A\mathbf{w} + B\mathbf{z} - \mathbf{c}$ in 2 are combined and the resulting ADMM updates become,

$$\begin{aligned} \mathbf{w}^{k+1} &= \underset{\mathbf{w}}{\text{argmin}}\ f(\mathbf{w}) + \frac{\rho}{2}\|A\mathbf{w} + B\mathbf{z}^k - \mathbf{c} + \mathbf{u}^k\|_2^2 \\ \mathbf{z}^{k+1} &= \underset{\mathbf{z}}{\text{argmin}}\ g(\mathbf{z}) + \frac{\rho}{2}\|A\mathbf{w}^{k+1} + B\mathbf{z} - \mathbf{c} + \mathbf{u}^k\|_2^2 \\ \mathbf{u}^{k+1} &= \mathbf{u}^k + (A\mathbf{w}^{k+1} + B\mathbf{z}^{k+1} - \mathbf{c}) \end{aligned} \tag{4}$$

[1]Note that, we use lowercase bold alphabets for representing vectors throughout the paper.

For the rest of the paper we shall use the scaled version of ADMM following [3].

Note that for a problem where the objective function can be decomposed as a sum of two functions ($f(\mathbf{w}), g(\mathbf{z})$ in eq. 1), ADMM provides a framework to solve two separate sub-problems ($\mathbf{w}$-step, $\mathbf{z}$-step of eq. 3) to obtain the final solution.

B. Consensus ADMM

Next we present a specific form called the consensus ADMM. This serves as a very useful approach to solve many problems in a distributed setting (as shown later in III for linear SVM). For this case, consider an optimization formulation where the $f(\mathbf{w})$ in eq. 1 can be decomposed into M independent parts i.e. $\sum_{t=1}^{M} f_t(\mathbf{w_t})$. Then the consensus ADMM can be written as,

$$\begin{aligned} \min_{\mathbf{w_1},\ldots,\mathbf{w_M},\mathbf{z}} \quad & f_1(\mathbf{w_1}) + \ldots + f_M(\mathbf{w_M}) + g(\mathbf{z}) \\ \text{subject to} \quad & A\mathbf{w}_1 + B\mathbf{z} = \mathbf{c} \\ & \vdots \\ & A\mathbf{w}_M + B\mathbf{z} = \mathbf{c} \end{aligned} \tag{5}$$

Note that, different from eq.1, here we solve M independent sub-problems. The equality constraint is called the *global consensus constraint* since it requires all the $\mathbf{w}_1 \ldots \mathbf{w}_M$ vectors to have a consensus with a global variable $\mathbf{z}$. This results in the following ADMM steps (in its scaled form) (see [3] for details),

$$\begin{aligned} \mathbf{w}_t^{k+1} &= \operatorname*{argmin}_{\mathbf{w_t}} f_t(\mathbf{w}_t) + \frac{\rho}{2}\|A\mathbf{w}_t + B\mathbf{z}^k - \mathbf{c} - \mathbf{u}_t^k\|_2^2 \\ \mathbf{z}^{k+1} &= \operatorname*{argmin}_{\mathbf{z}} g(\mathbf{z}) + \frac{\rho}{2}\sum \|A\mathbf{w}_t^{k+1} + B\mathbf{z} - \mathbf{c} + \mathbf{u}_t^k\|_2^2 \\ \mathbf{u}_t^{k+1} &= \mathbf{u}_t^k + A\mathbf{w}_t^{k+1} + B\mathbf{z}^{k+1} - \mathbf{c} \end{aligned} \tag{6}$$

Compared to eq 4, $\mathbf{w}_t$'s in eq 6 are updated independently and can be easily parallelized.

III. ADMM based Distributed Machine Learning Algorithms

In this section we discuss how we can utilize this ADMM framework to solve many machine learning algorithms. Under inductive settings a typical supervised machine learning problem involves estimating a function from noisy training samples $(\mathbf{x}_i, y_i)_{i=1}^N$, N= no. of training samples [30], [31]. There are two common types of supervised learning problems:-

- *Regression or real-valued function estimation*, $y = \hat{f}(\mathbf{x})$. In this case we have $y \in \Re$ and $x \in \Re^D$, D= dimension of the input space. The quality of prediction/estimation is measured by a user-defined loss function $L(\hat{f}_{\mathbf{w},b}(\mathbf{x}_i), y_i)$. Typical examples include, squared loss, ϵ-insensitive loss etc.
- *Classification or estimation of indicator function*, $y = \hat{f}(\mathbf{x})$. In this case we have $y \in \{+1, -1\}$ and $x \in \Re^D$, D= dimension of the input space. As before, the quality of prediction/estimation is measured by a user-defined loss function $L(\hat{f}_{\mathbf{w},b}(\mathbf{x}_i), y_i)$ like, logit loss, hinge loss, 0/1-loss etc.

A common optimization problem that is solved for both the supervised learning problems discussed above is:

$$\min_{\mathbf{w},b} \frac{1}{N}\sum_{i=1}^{N} L(\hat{f}_{\mathbf{w},b}(\mathbf{x}_i), y_i) + \lambda R(\mathbf{w}) \tag{7}$$

Here N is the total number of samples used to estimate the model ($\hat{f}_{\mathbf{w},b}$) parameterized by $\mathbf{w} \in \Re^D$ and $b \in \Re$. In this paper, we limit ourselves to linear parameterizations where, $\hat{f}_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}^\top\mathbf{x} + b$. L is a convex loss which measures the discrepancy between the model estimates and their true values/labels. R is a convex regularizer that penalizes the model complexity for better generalization on unseen future test samples.

In this paper we propose to solve a general class of optimization problem shown in eq. 7, and use this solver for many popular supervised machine learning algorithms (see Table I). We provide the ADMM updates for each of these algorithms and show that, at the heart of the ADMM updates for eq.7 is a QP problem during the $\mathbf{w}$-step which has the following form,

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^\top P\mathbf{w} - \mathbf{q}^\top\mathbf{w} \\ \text{subject to} \quad & l \preceq \mathbf{w} \preceq u \end{aligned} \tag{8}$$

We adopt the following strategies to solve this QP,

1) *Unconstrained case* (i.e. $l = -\infty, u = \infty$)
In this case we use a direct matrix inversion, $\mathbf{w}^* = P^{-1}\mathbf{q}$. Note that, for high-dimensional problems such matrix-inversion operations could become a bottleneck. However, more advanced QP solvers can be added in future versions of this work, e.g. ones based on conjugate gradient.
2) *Constrainined case* (i.e. l, u are finite)
We solve the QP problem using L-BFGS [32] method and apply warm-start strategy, i.e. initialize with $\mathbf{w}$ value from previous iteration.

Next we present the ADMM updates for the different ML algorithms in Table I.

A. L1/L2 Regression

In this sub-section, we consider the more generic elastic-net regularizer [33] with the least squares loss. The problem formulation is:-
Given input training data $(\mathbf{x}_i, y_i)_{i=1}^N$ with $\mathbf{x} \in \Re^D$ and $y \in \Re$, linear regression with elastic net regularization solves the

Table I
MACHINE LEARNING ALGORITHMS IN THE FORM OF EQ. 7

Methods	**Loss Functions** (with $\hat{f}_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$)	**Regularizer**
L1,L2,L1-L2 regularized linear regression	least-square: $\frac{1}{2N}\sum_i (y_i - b - \mathbf{x}_i^\top \mathbf{w})^2$	$\alpha\|\mathbf{w}\|_1 + (1-\alpha)\cdot\frac{1}{2}\|\mathbf{w}\|_2^2$ with, $\alpha \in [0,1]$
L1,L2,L1-L2 regularized logistic regression	logit loss: $\frac{1}{N}\sum_i \log(1+e^{-y_i(\mathbf{x}_i^\top \mathbf{w}+b)})$	
L1,L2,L1-L2 regularized linear SVM	hinge loss: $\frac{1}{N}\sum_i \left(1 - y_i(\mathbf{x}_i^\top \mathbf{w} + b)\right)_+$	
Group-Lasso	$\frac{1}{2N}\sum_i (y_i - b - \mathbf{x}_i^\top \mathbf{w})^2$	$\sum_{k=1}^G \sqrt{d_k}\left(\alpha\|\mathbf{w}_k\|_2 + (1-\alpha)\cdot\frac{1}{2}\|\mathbf{w}_k\|_2^2\right)$ G:= total groups, d_k:= size of the k^{th} group

following optimization:

$$\min_{\mathbf{w}} \lambda \sum_{j=1}^{D} \delta_j \left\{ \alpha \left|w_j\right| + (1-\alpha)\cdot \frac{{w_j}^2}{2} \right\} + \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - {\mathbf{x}_i}^\top \mathbf{w}\right)^2 \tag{9}$$

In this form we can include the intercept in the optimization problem by augmenting a column of ones to the input samples, i.e., $\hat{\mathbf{x}} = [\mathbf{x}, 1]_{(D+1)\times 1}$, and solving for $\hat{\mathbf{w}} = [\mathbf{w}, b]_{(D+1)\times 1}$.

Note that, the current form is more generic and can be easily adapted to solve both lasso ($\alpha = 1$) and ridge regression ($\alpha = 0$), in addition to elastic net [33]. Further, δ_j provides additional flexibility to this optimization problem,

- As discussed in [31], penalization of the intercept would make the algorithm depend on the origin chosen for y. Hence, we can avoid that by setting $\delta_{D+1} = 0$.
- In addition, we can incorporate apriori information to the penalization term. A special case is the group-lasso, where we set $\delta_j = \sqrt{d_k}$ for the k^{th} group of size d_k.

Here, the ADMM formulation is given as,

$$\min_{\mathbf{w},\mathbf{z}} \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - {\mathbf{x}_i}^\top \mathbf{w}\right)^2 + \lambda \sum_{j=1}^{D} \delta_j \left\{ \alpha \left|z_j\right| + (1-\alpha)\cdot \frac{{z_j}^2}{2} \right\} \tag{10}$$

subject to $\mathbf{w} - \mathbf{z} = \mathbf{0}$.

and the corresponding updates are,

$$\begin{aligned} \mathbf{w}^{k+1} &= \underset{\mathbf{w}}{\text{argmin}} \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - {\mathbf{x}_i}^\top \mathbf{w}\right)^2 \\ &+ \frac{\rho}{2} \parallel \mathbf{w} - \mathbf{z}^k + \mathbf{u}^k \parallel_2^2 \\ &= \underset{w}{\text{argmin}} \frac{1}{2}\mathbf{w}^\top P \mathbf{w} - \mathbf{q}^\top \mathbf{w} \\ &= P^{-1}\mathbf{q} \end{aligned} \tag{11}$$

$P = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^\top + \rho I_D$ and $\mathbf{q} = \frac{1}{N}\sum_{i=1}^{N} y_i \mathbf{x}_i + \rho(\mathbf{z}^k - \mathbf{u}^k)$

$$\begin{aligned} \mathbf{z}^{k+1} &= \underset{z}{\text{argmin}} \lambda \sum_j \delta_j \left\{ \alpha \left|z_j\right| + (1-\alpha)\cdot \frac{z_j^2}{2} \right\} \\ &+ \frac{\rho}{2}\|\mathbf{w}^{k+1} - \mathbf{z} + \mathbf{u}^k\|_2^2 \\ z_i^{k+1} &= \frac{c\kappa_j(w_j^{k+1} + u_j^k)}{1 + \lambda\delta_j(1-\alpha)/\rho} \end{aligned} \tag{12}$$

where $\kappa_j = \lambda\delta_j\alpha/\rho$ and $S_\kappa(t) = \left(1 - \frac{\kappa}{|t|}\right)_+ t = (t-\kappa)_+ - (-t-\kappa)_+$ is the soft-thresholding operator and,

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \mathbf{w}^{k+1} - \mathbf{z}^{k+1} \tag{13}$$

As seen above, for big-data problems $\mathbf{w}$-step poses as the main bottleneck. This however can be scaled through distributed computation of $\sum_i \mathbf{x}_i \mathbf{x}_i^T$ and $\sum_i y_i \mathbf{x}_i$. Finally, the $\mathbf{w}$-update is transformed to a matrix inversion problem as shown in eq. 11. The $\mathbf{z}, \mathbf{u}$ - updates can be easily obtained as shown in eq. 12 and 13.

B. Group-Lasso

Next we consider a very specific method called the Group Lasso. In this case we assume that the apriori grouping information is available to form a composite weight vector of G groups, denoted as $\mathbf{w} = [\underbrace{w_1^{(1)} \cdots w_{d_1}^{(1)}}_{group\ 1}, \ldots, \underbrace{w_1^{(g)} \cdots w_{d_g}^{(g)}}_{group\ g}, \ldots, w^{(G)}]$, where $w_k^{(g)}$=k^{th} feature of the g^{th} group, $w^{(G)} = b$ (the intercept), d_g = size of the g^{th} group. Then the group-lasso regularized linear regression model is given by,

$$\min_{\mathbf{w}} \frac{1}{2N}\sum_{i=1}^{N}(y_i - \mathbf{x}_i^\top \mathbf{w})^2 + \lambda \sum_{g=1}^{G} \delta_g \left\{ \alpha\|w_g\|_2 + (1-\alpha)\cdot\frac{1}{2}\|w_g\|_2^2 \right\} \tag{14}$$

In practice, we use $\delta_g = \sqrt{d_g}$ and $\delta_G = 0$ for the intercept. Following the same procedure as above, we get the $\mathbf{w}$-update and $\mathbf{u}$-update which are exactly the same as in the elastic-net regularized case, the only difference is the

$\mathbf{z}$-update which is given as,

$$\mathbf{z}_g^{k+1} = \frac{S_{\kappa_g}(w_g^{k+1} + u_g^k)}{1 + \lambda\delta_g(1-\alpha)/\rho} \tag{15}$$

where $\kappa_g = \lambda\delta_g\alpha/\rho$, and S_κ is the block soft-thresholding operator $S_\kappa(t) = \left(1 - \frac{\kappa}{\|t\|_2}\right)_+ t$

C. L1/L2-Logistic Regression

In this sub-section, we switch towards classification problems. Specifically, we consider the logistic regression classification method. Given input training data $(\mathbf{x}_i, y_i)_{i=1}^n$ with $\mathbf{x} \in \Re^D$ and $y \in \{-1, +1\}$, the logistic regression model is estimated by solving the following optimization problem:

$$\min_{\mathbf{w}} \quad \tfrac{1}{N}\textstyle\sum_{i=1}^N \log(1 + e^{-y_i\mathbf{x}_i^\top \mathbf{w}}) \tag{16}$$

$$+\lambda \sum_{j=1}^{D} \delta_j \left\{\alpha\,|w_j| + (1-\alpha)\cdot\tfrac{w_j^2}{2}\right\} \tag{17}$$

The corresponding ADMM form is as follows:

$$\min_{\mathbf{w},\mathbf{z}} \quad \tfrac{1}{N}\textstyle\sum_{i=1}^N \log(1 + e^{-y_i\mathbf{x}_i^\top \mathbf{w}}) \tag{18}$$

$$+\lambda \sum_{j=1}^{D} \delta_j \left\{\alpha\,|z_j| + (1-\alpha)\cdot\tfrac{{z_j}^2}{2}\right\}$$

subject to $\mathbf{w} - \mathbf{z} = 0$

Same as before we use $\hat{\mathbf{x}} = [\mathbf{x}, 1]_{(D+1)\times 1}$, with $\hat{\mathbf{w}} = [\mathbf{w}, b]_{(D+1)\times 1}$. The resulting ADMM updates are,

$$\begin{aligned}
\mathbf{w}^{k+1} &= \operatorname*{argmin}_{\mathbf{w}} \frac{1}{N}\sum_i \log(1 + e^{-y_i\mathbf{x}_i^\top \mathbf{w}}) \qquad (19)\\
&+ \frac{\rho}{2}\|\mathbf{w} - \mathbf{z}^k + \mathbf{u}^k\|_2^2\\
\mathbf{z}^{k+1} &= \operatorname*{argmin}_{\mathbf{z}} \lambda \sum_j \delta_j \left\{\alpha\,|z_j| + (1-\alpha)\cdot\frac{z_j^2}{2}\right\}\\
&+ \frac{\rho}{2}\|\mathbf{w}^{k+1} - \mathbf{z} + \mathbf{u}^k\|_2^2\\
\mathbf{u}^{k+1} &= \mathbf{w}^{k+1} - \mathbf{z}^{k+1} + \mathbf{u}^k
\end{aligned}$$

Note that, the $\mathbf{z}$ and $\mathbf{u}$ updates are same as in eq 12 and 13 respectively. For the w-step we use Newton updates given below. Let,

$$\begin{aligned}
l(\mathbf{w}) &= \frac{1}{N}\sum_i \log(1 + e^{-y_i\mathbf{x}_i^\top \mathbf{w}})\\
&+ \frac{\rho}{2}\|\mathbf{w} - \mathbf{z}^k + \mathbf{u}^k\|_2^2
\end{aligned}$$

then,

$$\triangledown_{\mathbf{w}} l(\mathbf{w}) = -\frac{1}{N}\sum_i y_i(1-p_i)\mathbf{x}_i + \rho(\mathbf{w} - \mathbf{z}^k + \mathbf{u}^k),$$

$$\triangledown^2_{\mathbf{w}} l(\mathbf{w}) = \frac{1}{N}\sum_i p_i(1-p_i)\mathbf{x}_i\mathbf{x}_i^\top + \rho I$$

where, $p_i = 1/(1 + e^{-\mathbf{w}^\top \mathbf{x}})$. Hence, the optimal $\mathbf{w}^{k+1}$ can be obtained through the iterative Algorithm 1

Algorithm 1: Iterative Algorithm for $\mathbf{w}^{k+1}$

```
Input: $\mathbf{w}^k$, $\mathbf{z}^k$, $\mathbf{u}^k$
Output: $\mathbf{w}^{k+1}$
initialize $\mathbf{v}^{(0)} \leftarrow \mathbf{w}^k$, $j \leftarrow 0$ ;
while not converged do
    $p_i^{(j)} \leftarrow 1/(1 + e^{-\mathbf{x}_i^\top \mathbf{v}^{(j)}})$;
    $P^{(j)} \leftarrow \frac{1}{N}\sum_i p_i^{(j)}(1 - p_i^{(j)})\mathbf{x}_i\mathbf{x}_i^\top + \rho I$ ;
    $\mathbf{q}^{(j)} \leftarrow -\frac{1}{N}\sum_i y_i(1-p_i^{(j)})\mathbf{x}_i + \rho(\mathbf{v}^{(j-1)} - \mathbf{z}^k + \mathbf{u}^k)$;
    $\mathbf{v}^{(j+1)} \leftarrow \mathbf{v}^{(j)} - \boxed{(P^{(j)})^{-1}\mathbf{q}^{(j)}}$ (distributed) ;
    $j \leftarrow j + 1$;
return $\mathbf{w}^{k+1} \leftarrow \mathbf{v}^{(j)}$;
```

D. Linear SVM

Finally we show how to use this similar framework to solve Linear SVM. Note that, a detailed analysis for distributed SVM using ADMM has already been shown in [15]. However, even though the technicalities are similar,we solve a slightly different problem (hinge loss + elastic net), and show it for completeness. Finally, different from [15], [34], we rather use an L-BFGS approach to solve each sub-problem as discussed next.

The SVM-problem formulation is provided next, Given input training data $(\mathbf{x}_i, y_i)_{i=1}^n$ with $\mathbf{x} \in \Re^D$ and $y \in \{-1, +1\}$, the elastic-net regularized linear SVM solves the following optimization problem:

$$\min_{\mathbf{w}} \quad \tfrac{C}{N}\textstyle\sum_{i=1}^N (1 - y_i\mathbf{x}_i^\top \mathbf{w})_+ \tag{20}$$

$$+ \sum_{j=1}^{D} \delta_j \left\{\alpha|w_j| + (1-\alpha)\cdot\tfrac{w_j^2}{2}\right\}$$

As again, $\alpha \in [0, 1]$ controls the effect of L1 vs. L2 regularization and $\delta_D = 0$ is used to avoid regularization in intercept space. Here, unlike the previous models the hinge loss in SVM is nonsmooth. To tackle this issue, we use the consensus ADMM and rather solve smaller SVM-like sub-problems as also shown in [15]. The advantage to this approach is that each smaller SVM-like sub-problem can be now solved in the dual space using a QP solver. This is shown next.

The consensus ADMM formulation for the problem is,

$$\min_{\mathbf{w}_1,\ldots,\mathbf{w}_M,\mathbf{z}} \quad \tfrac{C}{N}\textstyle\sum_{t=1}^M \sum_{i\in B_t}(1 - y_i\mathbf{x}_i^\top \mathbf{w}_t)_+$$

$$+ \sum_j \delta_j \left\{\alpha\,|z_j| + (1-\alpha)\cdot\tfrac{z_j^2}{2}\right\} \tag{21}$$

$$\text{subject to} \quad \mathbf{w}_t - \mathbf{z} = 0, \quad t = 1, \ldots, M$$

and the corresponding updates are,

$$\begin{aligned}
\mathbf{w}_t^{k+1} &= \underset{\mathbf{w}_t}{\text{argmin}} \; \frac{C}{N}\sum_{i\in B_t}(1-y_i\mathbf{x}_i^\top \mathbf{w}_t)_+ \\
&+ \frac{\rho}{2}\|\mathbf{w}_t - \mathbf{z}^k + \mathbf{u}_t^k\|_2^2, \quad t = 1,\dots,M \qquad (22)\\
\mathbf{z}_j^{k+1} &= \frac{S_{\kappa_j}\left(\frac{1}{M}\sum_{t=1}^M (w_{tj}^{k+1} + u_{tj}^k)\right)}{1+\lambda\delta_j(1-\alpha)/\rho}, j = 1,\dots,D \\
\mathbf{u}_t^{k+1} &= \mathbf{u}_t^k + \mathbf{w}_t^{k+1} - \mathbf{z}^{k+1}, \quad t = 1,\dots,M
\end{aligned}$$

Note that now the $\mathbf{w}$-update is an SVM like problem on a subset B_t. This can be solved in the dual form as shown next.

For each subset B_t,

$$\begin{aligned}
\mathbf{w}_t^{k+1} &= \underset{\mathbf{w}_t}{\text{argmin}} \; \frac{C}{N}\sum_{i\in B_t}\xi_i + \frac{\rho}{2}\|\mathbf{w}_t - \mathbf{z}^k + \mathbf{u}_t^k\|_2^2 \\
&\text{s.t. } y_i\mathbf{x}_i^\top \mathbf{w} \geq 1-\xi_i, \; \xi_i \geq 0, \; i \in B_t \qquad (23)
\end{aligned}$$

This transforms to the following QP (with constraints) given below,

$$\begin{aligned}
\min_{\boldsymbol{\alpha}} \quad & \tfrac{1}{2}\boldsymbol{\alpha}^\top P\boldsymbol{\alpha} + \mathbf{q}^\top\boldsymbol{\alpha} \qquad (24)\\
\text{s.t.} \quad & 0 \leq \alpha_i \leq C/N, \quad i \in B_t
\end{aligned}$$

with,

$$\begin{aligned}
P_{ij} &= y_iy_j\mathbf{x}_i^\top\mathbf{x}_j \\
\mathbf{q}_i &= y_i\mathbf{x}_i^\top(\mathbf{z}^k - \mathbf{u}_t^k) - 1
\end{aligned}$$

We use L-BFGS to solve the above QP and finally obtain,

$$\mathbf{w}^{k+1} = \frac{1}{\rho}\sum_{i\in B_t}\alpha_i^{k+1}y_i\mathbf{x}_i + \mathbf{z}^k - \mathbf{u}_t^k \qquad (25)$$

as the final SVM solution. This can also be used to accommodate for non-linear SVM following [35].

IV. Experiments and Results

Next we provide the performance comparison of our implemented algorithms with the publicly available MLLIB library packaged with Apache Spark 3.0.

A. System Configuration

The Hadoop cluster configuration for our experiments is provided below,

- No. of Nodes = 6 (Hadoop Version - Apache 1.1.1)
- No. of cores (per node) = 12 core (Intel Xeon @ 3.20GHz)
- RAM size (per node) = 32 GB
- Hard Disk size (per node) = 500 GB

For implementation we use the python interface (pyspark) already available in [28]. Further, our spark framework has been configured based on the recommendations available at [36]. i.e.

- spark.num.executors = 17
- spark.executor.memory = 6 GB
- spark.driver.memory = 4 GB
- spark.driver.maxResultSize = 4 GB

B. Datasets

We generate synthetic datasets of different sizes for our experiments. The datasets are generated to capture the sparsity as well as the grouping behavior of the different methods. The dataset used for the classification methods is described below,

Dataset for Classification Problems: In this case $\mathbf{x} \in \Re^D$ is generated from a multivariate normal distribution $\mathcal{N}(\mathbf{0}, \Sigma)$. Here,

$$\text{Correlation Matrix}, \Sigma = \underbrace{\begin{pmatrix} \underbrace{\begin{matrix} 1 & 0.2 & 0.2 \\ 0.2 & \ddots & 0.2 \\ 0.2 & 0.2 & 1 \end{matrix}}_{10} & \mathbf{0} \\ \mathbf{0} & \ddots \end{pmatrix}}_{G*10}$$

is block diagonal, and controls the grouping properties of the problem. We fix the number of variables per group to 10. The pairwise correlation *within* each group is 0.2, and that *between* each group is 0. The y - value (class label) is generated as shown below,

$$y = \text{sign}(\mathbf{w}^\top\mathbf{x} + \varepsilon) \qquad (26)$$

where, $\mathbf{w}$ controls the sparsity of the problem. For this paper we set the sparsity parameter to 0.8, i.e. 80% of the groups have zero weight vector. For the remaining 20%, the weight vectors are alternated between +1/-1. i.e.

$$\begin{aligned}
\mathbf{w} = & [\underbrace{1,1,1,1,1,-1,-1,-1,-1,-1}_{group\ 1},\dots \\
& \dots\underbrace{1,1,1,1,1,-1,-1,-1,-1,-1}_{group\ g},\dots \\
& \dots \quad \underbrace{0,0,0,\dots}_{\text{remaining 80 \% sparse groups}} \quad]
\end{aligned}$$

Further we add a gaussian noise to the model $\varepsilon \sim \mathcal{N}(0,1)$. The above settings are used to generate two separate data of different sizes (shown below),

- No. of training samples (N) = 2000000 and Dimension of each samples (D) = 100,
- No. of training samples (N) = 20000000 and Dimension of each samples (D) = 100

The generated data is saved in a comma separated format, which takes upto (approx.) 5 GB and 50 GB of disk space respectively.

Dataset for Regression Problems: The generation of this

Table II
COMPUTATION TIME COMPARISON BETWEEN ADMM VS. MLLIB (IN SEC) FOR CLASSIFICATION METHODS

Methods	**ADMM**	**MLLIB**
Data Set size = 5 GB with N = 2000000, D = 100		
L2- logistic regression ($\lambda = 0.1, \alpha = 0$)	157.57 (0.04)	139.68 (2.06)
L1- logistic regression ($\lambda = 0.1, \alpha = 1$)	157.05 (1.54)	266.9 (169.16)
L1+L2- logistic regression ($\lambda = 0.1, \alpha = 0.5$)	155.2(1.23)	Not available
Data Set size = 50 GB with N = 20000000, D = 100		
L2- logistic regression ($\lambda = 0.1, \alpha = 0$)	13937.3 (10.34)	14045.7 (411.78)
L1- logistic regression ($\lambda = 0.1, \alpha = 1$)	15381.8 (5.59)	13155.2 (307.60)
L1+L2- logistic regression ($\lambda = 0.1, \alpha = 0.5$)	15472.1 (13.25)	Not available

data follows exactly the same as in classification problems; except that the y-values are generated as below,

$$y = \mathbf{w}^\top \mathbf{x} + 2 + \varepsilon \tag{27}$$

As before we use two separate data of different sizes,

- No. of training samples (N) = 2000000 and Dimension of each samples (D) = 100,
- No. of training samples (N) = 20000000 and Dimension of each samples (D) = 100

C. Results

Here we provide comparison of the computation times for our ADMM implementation vs. MLLIB for both classification and regression problems. In general, the computation time of the ADMM based methods depend heavily on a number of parameters like , ρ - update (see [3], [13]), convergence criteria etc. For simplicity, we follow the ρ-update suggested in (eq 3.13 of [3]). Further, our current stopping criteria dictates convergence of the solution, when the primal and dual residual conjointly goes below a tolerance value of 10^{-3} (following [3]). On the other hand, MLLIB does not provide any control on the convergence criteria. Hence for our experiments we keep the default settings. Tables II amd III provides the average computation times (in seconds) over three runs of the experiment for the classification and regression problems respectively. The standard deviation are provided in parenthesis. In the current version of the paper our results are limited to L1/L2-Logistic and L1/L2-Linear regression. Additional results for L1/L2-SVM and Group Lasso shall be provided in a extended version of the paper.

Based on our results in Table II and III, the ADMM implementation performs similar to the MLLIB in terms of computation speed except for the regression problem [2]. For the regression problem we report the computation time for one iteration of the ADMM updates. This approximate solution still outperfomed the MLLIB's solution in terms of accuracy. Hence, the current ADMM based framework provides as a viable alternative to the SGD based approach implemented in MLLIB. In addition, this framework supports a wide range of scalable ML algorithms, which can prove as an useful arsenal for data-scientists to tackle big-data problems.

V. CONCLUSION

In this paper we present a generic convex optimization problem for most ML algorithms. We identify ADMM as a viable approach to solve this generic convex optimization problem, and derive the ADMM updates specific to each ML algorithms (listed in Table I). The current paper provides the update steps for linear parameterization. However, it can be easily extended to non-linear cases following [14]. As shown in section III, at the heart of the ADMM updates lies a QP which can be solved in a distributed fashion. Our results show that this ADMM based approach performs similar in comparison to the publicly available MLLIB in terms of computation speed. This presents ADMM as a viable alternative to MLLIB for big-data problems, with the added advantage of more machine learning algorithms.

Finally, we note that the current implementation is limited by the dimension of the problem; as it needs to solve a QP in the $\mathbf{w}$ - update. This motivates the need for future

[2]The MLLIB package distributed with Spark 1.3 provides incorrect implementation of the original Logistic Regression algorithm. A correction has been made in the latest Spark 1.4 version (see [37]). This has not been included in this paper. However, the Spark 1.3 's implementation can still be considered as an approximate comparison representative of the SGD approach. Further, the convergence criteria for MLLIB cannot be controlled. In terms of accuracy, for both classification and regression problems, the MLLIB tool provided sub-optimal solutions.

Table III
COMPUTATION TIME COMPARISON BETWEEN ADMM VS. MLLIB (IN SEC) FOR REGRESSION METHODS

Methods	**ADMM**	**MLLIB**
Data Set size = 5 GB with N = 2000000, D = 100		
L2- linear regression ($\lambda = 0.1, \alpha = 0$)	425.09 (17.26)	429.83(190.02)
L1- linear regression ($\lambda = 0.1, \alpha = 1$)	416.95 (3.5)	444.79 (210.22)
L1+L2- linear regression ($\lambda = 0.1, \alpha = 0.5$)	409.50(2.5)	Not available
Data Set size = 50 GB with N = 20000000, D = 100		
L2- linear regression ($\lambda = 0.1, \alpha = 0$)	4209.95(10.5)	29244.43(100.12)
L1- linear regression ($\lambda = 0.1, \alpha = 1$)	4233.13 (6.89)	23526.45(200.23)
L1+L2- linear regression ($\lambda = 0.1, \alpha = 0.5$)	4150.81 (10.25)	Not available

research towards scalable options for the QP problem. In addition to that, there has been a gamut of research towards newer ADMM update strategies for faster convergence of the algorithm [3], [13]. These advanced strategies have not been included in this version of the paper and can be extended as future work.

ACKNOWLEDGMENT

The authors would like to thank Juergen Heit from Research and Technology Center, Robert Bosch LLC; for multiple discussions on the Spark configurations for the experimental settings. They would also like to thank Max Rizvanov for his support with the Hadoop cluster management.

REFERENCES

[1] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2011, pp. 693–701.

[2] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2010, pp. 2595–2603.

[3] 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, no. 1, pp. 1–122, 2011.

[4] D. Gabay and B. Mercier, "A dual algorithm for the solution of nonlinear variational problems via finite element approximation," *Computers & Mathematics with Applications*, vol. 2, no. 1, pp. 17–40, 1976.

[5] T. Goldstein, B. ODonoghue, and S. Setzer, "Fast alternating direction optimization methods," *CAM report*, pp. 12–35, 2012.

[6] R. Glowinski and A. Marroco, "Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires," *ESAIM: Mathematical Modelling and Numerical Analysis-Modélisation Mathématique et Analyse Numérique*, vol. 9, no. R2, pp. 41–76, 1975.

[7] D. Mahajan, S. S. Keerthi, S. Sundararajan, and L. Bottou, "A functional approximation based distributed learning algorithm," *arXiv preprint arXiv:1310.8418*, 2013.

[8] O. Shamir, N. Srebro, and T. Zhang, "Communication efficient distributed optimization using an approximate newton-type method," *arXiv preprint arXiv:1312.7853*, 2013.

[9] C. H. Teo, S. Vishwanthan, A. J. Smola, and Q. V. Le, "Bundle methods for regularized risk minimization," *The Journal of Machine Learning Research*, vol. 11, pp. 311–365, 2010.

[10] X. Zhang, "Probabilistic methods for distributed learning," Ph.D. dissertation, Duke University, 2014.

[11] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," in *Advances in Neural Information Processing Systems*, 2011, pp. 873–881.

[12] X. Meng, J. K. Bradley, B. Yavuz, E. R. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, "Mllib: Machine learning in apache spark," *CoRR*, vol. abs/1505.06807, 2015. [Online]. Available: http://arxiv.org/abs/1505.06807

[13] R. Nishihara, L. Lessard, B. Recht, A. Packard, and M. I. Jordan, "A General Analysis of the Convergence of ADMM," *ArXiv e-prints*, Feb. 2015.

[14] V. Sindhwani and H. Avron, "High-performance Kernel Machines with Implicit Distributed Optimization and Randomization," *ArXiv e-prints*, Sep. 2014.

[15] 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*, 2012, pp. 1398–1406.

[16] MATLAB, *version 8.5 (R2015a)*. Natick, Massachusetts: The MathWorks Inc., 2015.

[17] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013. [Online]. Available: http://www.R-project.org/

[18] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, and B. Wiswedel, “KNIME: The Konstanz Information Miner,” in *Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007)*. Springer, 2007.

[19] “Rapidminer,” https://rapidminer.com/, accessed: 2015-06-30.

[20] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA data mining software: An update,” *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009. [Online]. Available: http://www.sigkdd.org/explorations/issues/11-1-2009-07/p2V11n1.pdf

[21] “Revolution r,” http://www.revolutionanalytics.com/revolution-r-enterprise, accessed: 2015-06-30.

[22] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, “Graphlab: A new parallel framework for machine learning,” in *Conference on Uncertainty in Artificial Intelligence (UAI)*, Catalina Island, California, July 2010.

[23] “Oracle data miner,” http://www.oracle.com, accessed: 2015-06-30.

[24] “Hp vertica,” http://www.vertica.com/, accessed: 2015-06-30.

[25] “Pivotal,” http://pivotal.io/, accessed: 2015-06-30.

[26] “Revolution analytics rhadoop,” https://github.com/RevolutionAnalytics/RHadoop/wiki, accessed: 2015-06-30.

[27] Apache Software Foundation. Apache mahout:: Scalable machine-learning and data-mining library. [Online]. Available: http://mahout.apache.org

[28] “Apache spark,” https://spark.apache.org/, accessed: 2015-06-30.

[29] “Alpine data labs,” http://alpinenow.com/, accessed: 2015-06-30.

[30] V. Cherkassky and F. M. Mulier, *Learning from Data: Concepts, Theory, and Methods*. Wiley-IEEE Press, 2007.

[31] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[32] D. P. Bertsekas, *Nonlinear Programming*. Belmont, MA: Athena Scientific, 1999.

[33] H. Zou and T. Hastie, “Regularization and variable selection via the elastic net,” *Journal of the Royal Statistical Society, Series B*, vol. 67, pp. 301–320, 2005.

[34] C.-Y. Lin, C.-H. Tsai, C.-P. Lee, and C.-J. Lin, “Large-scale logistic regression and linear support vector machines using spark,” in *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 2014, pp. 519–528.

[35] A. Rahimi and B. Recht, “Random features for large-scale kernel machines,” in *Advances in neural information processing systems*, 2007, pp. 1177–1184.

[36] “How-to: Tune your apache spark jobs (part 2),” http://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-2/, accessed: 2015-06-30.

[37] “Mllib (spark) question.” https://www.mail-archive.com/user@spark.apache.org/msg32244.html, accessed: 2015-06-30.