Scalable Machine Learning on Spark for Multiclass Problems

Goutham Kamath[1*], Sauptik Dhar[2], Naveen Ramakrishnan[2], David Hallac[3], Jure Leskovec[3], Mohak Shah[2,4]
[1]Georgia State University, GA [2]Robert Bosch Research and Technology Center, CA [3]Stanford University, CA [4]University of Illinois at Chicago, IL
Emails: gkamath1@student.gsu.edu {jure, hallac}@cs.stanford.edu {sauptik.dhar, naveen.ramakrishnan, mohak.shah}@us.bosch.com

Introduction and Motivation

- Big-Data capability provides advantage by harnessing sophisticated insights relevant to various real-life applications.
- Spark's distributed in-memory computing framework provides practitioners with a *powerful*, *fast*, *scalable* and *easy* way to build big-data ML algorithms.
- However, existing large-scale ML tools on Spark such as : MLlib [1] or PhotonML [2] provide limited coverage for multi-class problems and sometimes inaccurate solutions.
- This work extends " *ADMM based scalable machine learning on Spark* " [3] to handle large-scale machine learning for multi-class problems.

Existing state-of-the-art technologies

Distributed Algorithms

First Order methods: use first order gradient estimates, secant approximates etc.

- Low per-iteration computation complexity,
- Dimension independent convergence,

However,

- Slower convergence due to oscillations,
- Not well suited for ill-conditioned problems typically seen in Machine Learning, E.g. Parallel SGD [4], Hogwild! [5], Splash [6].

Second Order methods: use additional second order Hessian (approximate) information.

- Captures the curvature of the objective function,
- Faster convergence rate than first order methods,
- Well suited for ill conditioned problems seen in Machine Learning,

However,

- Do not scale favorable with dimension,
- High per-iteration computation complexity. E.g. ADMM [7], DANE[8] etc.

Randomized Algorithms: uses a subsampled or low rank representation of the original big-data to solve a small-scale equivalent problem.

- Can be solved using traditional ML software,
- Avoids the need for distributed storage/analytics systems.

However,

- Inexact/approximate solutions depends on data property. E.g. LOCO [9]

- We propose a generic multi-class formulation and adopt the second order Alternating Direction Method of Multipliers (ADMM) to obtain more accurate solutions without compromising on computational efficiency.

Our Approach

Generic Multiclass Formulation

Given training samples $T := (\mathbf{x}_i, y_i)_{i=1}^N$ where $\mathbf{x} \in \mathbb{R}^D$ and $y \in \{1,\ldots,C\}$

$N :=$ no. of observations (samples) $D :=$ dimensions. $C :=$ Total classes

Solve, $\min_{\mathbf{w}_1 \ldots \mathbf{w}_C} \frac{1}{N}\sum_{i=1}^N L(f_{\mathbf{w}_1 \ldots \mathbf{w}_C}(\mathbf{x}_i), y_i) + \lambda R(\mathbf{w}_1 \ldots \mathbf{w}_C)$

$$\equiv \min_{\mathbf{w}_1 \ldots \mathbf{w}_C, \mathbf{z}_1 \ldots \mathbf{z}_C} \frac{1}{N}\sum_{i=1}^N L(f_{\mathbf{w}_1 \ldots \mathbf{w}_C}(\mathbf{x}_i), y_i) + \lambda R(\mathbf{z}_1 \ldots \mathbf{z}_C) \quad s.t. \quad \mathbf{w}_c = \mathbf{z}_c \quad \forall c \in \{1 \ldots C\}$$

where, $L(\cdot) :=$ loss function, $R(\cdot) :=$ regularization.

Multinomial Logistic Regression ADMM Step (k+1th iteration):

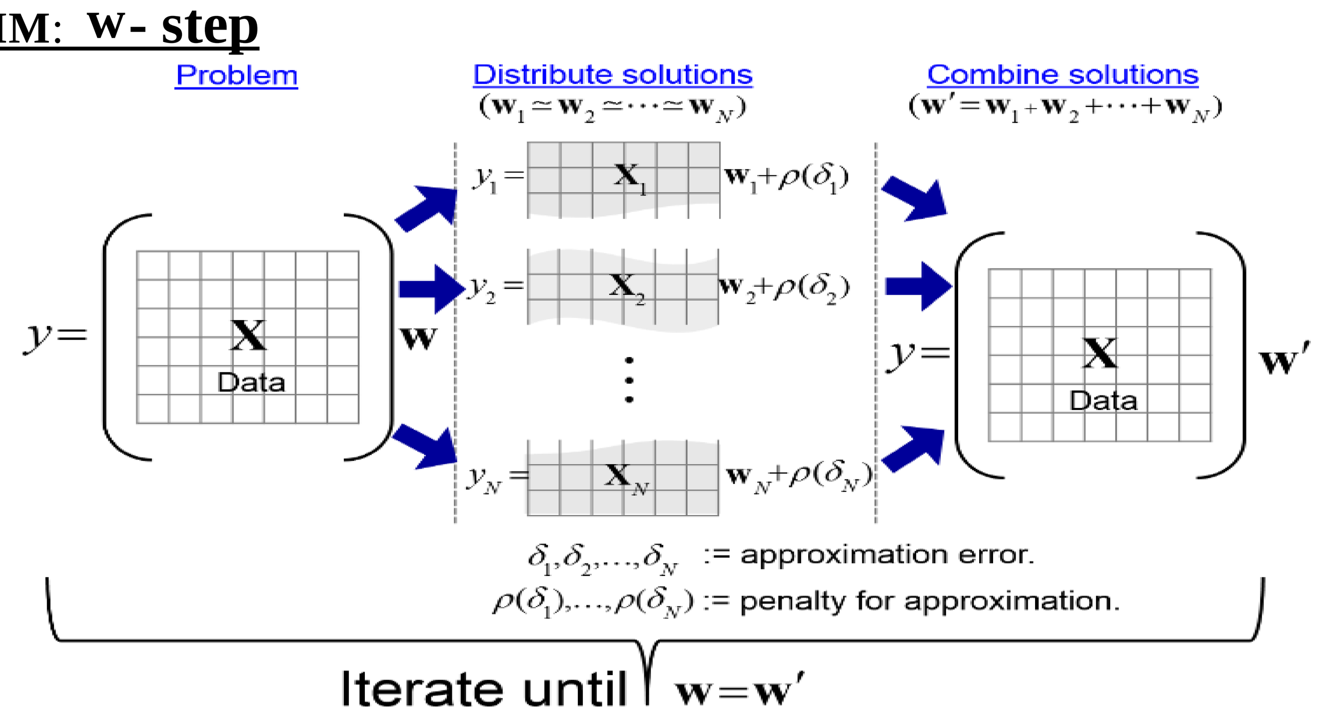
- $(\mathbf{w}_1, \ldots \mathbf{w}_C)^{k+1} = \arg\min_{\mathbf{w}_1,\ldots \mathbf{w}_C} \frac{1}{N}\sum_{i=1}^N \log\left(\frac{e^{\mathbf{x}_i^T \mathbf{w}_{y_i}}}{\sum_{c=1}^C e^{\mathbf{x}_i^T \mathbf{w}_c}}\right) + \frac{\rho}{2}\sum_{c=1}^C \left\|\mathbf{w}_c - \mathbf{z}_c^k + \mathbf{u}_c^k\right\|_2^2$
- $(\mathbf{z}_1, \ldots \mathbf{z}_C)^{k+1} = \arg\min_{\mathbf{z}_1,\ldots \mathbf{z}_C} \lambda \sum_{c,j} \delta_{jc}\{\alpha |z_{jc}| + (1-\alpha)\frac{z_{jc}^2}{2}\} + \frac{\rho}{2}\sum_{c=1}^C \left\|\mathbf{w}_c^{k+1} - \mathbf{z} + \mathbf{u}_c^k\right\|_2^2$
- $\mathbf{u}_c^{k+1} = \mathbf{w}_c^{k+1} - \mathbf{z}_c^{k+1} + \mathbf{u}_c^k \quad \forall c$

Here, $L(f_{\mathbf{w}_1 \ldots \mathbf{w}_C}(\mathbf{x}_i), y_i) = \sum_{i=1}^N \log\left(\frac{e^{\mathbf{x}_i^T \mathbf{w}_{y_i}}}{\sum_{c=1}^C e^{\mathbf{x}_i^T \mathbf{w}_c}}\right)$ and $R(\mathbf{z}_1 \ldots \mathbf{z}_C) = \sum_{c,j} \delta_{jc}\{\alpha |z_{jc}| + (1-\alpha)\frac{z_{jc}^2}{2}\}$

Remarks

- ADMM decomposes a larger problem into two (or many) smaller sub-problems in variables $\mathbf{w}, \mathbf{z}$
- Computation overhead due to big-data is handled in the $\mathbf{w}$-step in a distributed fashion,
- Many multiclass ML algorithms can be solved using a simple distributed QP solver.

ADMM: w- step



* Work done during Goutham Kamath's internship at Bosch.

Supported Multiclass Algorithms

Methods	Loss Function $L(f_{\mathbf{w},b}(\mathbf{x}_i), y_i)$	Regularizer $R(\mathbf{w})$
Multiclass Classification		
L1, L2, L1-L2 regularized multinomial regression	$1/N \sum_i \log(e^{\mathbf{x}_i^T \mathbf{w}_{y_i}} / \sum_{c=1}^C e^{\mathbf{x}_i^T \mathbf{w}_c})$	$\sum_{c=1}^C \sum_{j=1}^D \delta_{jc}\left\{\alpha\|\mathbf{w}_{jc}\| + (1-\alpha)\frac{\mathbf{w}_{jc}^2}{2}\right\}$ with $\alpha \in [0,1]$
L1, L2, L1-L2 SVM	$1/N \sum_i ((1-\delta_{y_i c}) - \mathbf{x}_i^T(\mathbf{w}_{y_i} - \mathbf{w}_c))_+$	

Experimental Results

Hadoop Configuration (Apache 1.1.1)

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

Spark Configuration (Apache 1.6)

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

ADMM Configuration: Adaptive ρ update [7] with $\rho_{initial} = 0.5$ and $\delta = 1, \lambda = 1$

Synthetic Data Set (Binary Classification used in [3])

$\mathbf{x} \in \mathbb{R}^D \quad \varepsilon \sim \mathbb{N}(\mathbf{0}, \Sigma) \quad y = sign(\mathbf{w}^T \mathbf{x} + \varepsilon)$

$$\mathbf{w} = [\underbrace{1, 1, 1, 1, 1, -1, -1, -1, -1, -1}_{group\ 1}, \ldots, \underbrace{1, 1, 1, 1, 1, -1, -1, -1, -1, -1}_{group\ g}, \ldots, \underbrace{0, 0, 0, \ldots}_{\text{remaining 80\% sparse group}}] \qquad \Sigma = \begin{pmatrix} \underbrace{\begin{matrix} 1 & 0.2 & 0.2 \\ 0.2 & \ddots & 0.2 \\ 0.2 & 0.2 & 1 \end{matrix}}_{10} & 0 \\ 0 & \ddots \end{pmatrix}$$

Small Data (~5GB) , N = 200000
Big Data (~50GB), N = 2000000
Dimension (D) = 100

Table 1. Average time performance over 10 experiments in sec (std. deviation shown in parenthesis).

Methods	ADMM	MLlib
Data Set size = 5GB with N = 200000, D =100		
L2-logistic regression ($\lambda = 0.1 \quad \alpha = 0$)	157.57(0.04)	139.68(2.06)
L1-logistic regression ($\lambda = 0.1 \quad \alpha = 1$)	157.05(1.54)	266.9(169.16)
Data Set size = 50GB with N = 2000000, D =100		
L2-logistic regression ($\lambda = 0.1 \quad \alpha = 0$)	13937.3 (10.34)	14045.7 (411.78)
L1-logistic regression ($\lambda = 0.1 \quad \alpha = 1$)	15381.8 (5.59)	13155.2 (307.60)

Real-Life MNIST Handwritten Digit Recognition Data(~ 1GB .csv format)

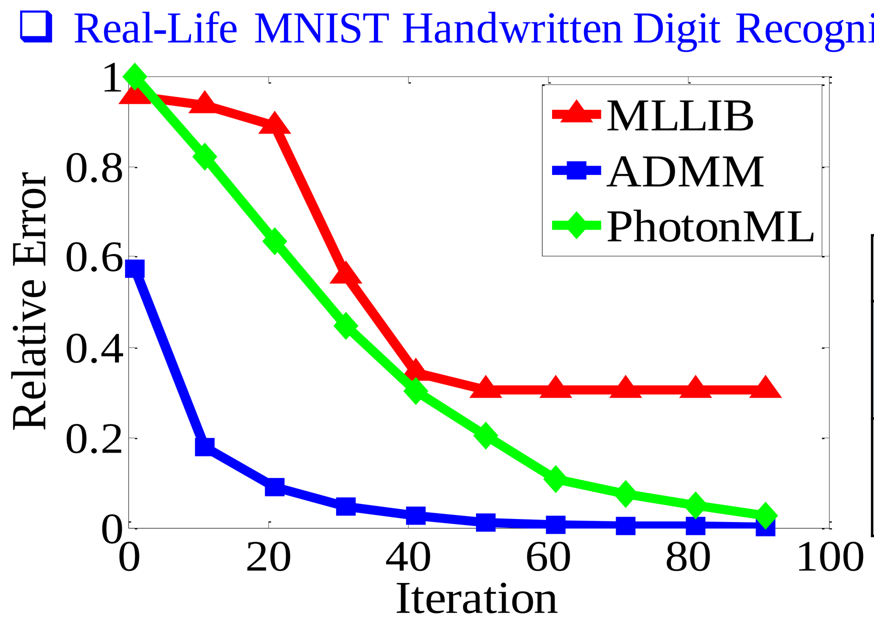


Fig. 1. Solution Convergence

Table 2. Time performance (in sec)

Methods	ADMM	PhotonML
Binary (5 vs. 8)	321.54 (12.52)	118.81 (1.39)
Multiclass (3 vs 5 vs 8)	1229.56 (50.44)	N/A

Discussion

- Fig. 1 shows the relative error of the estimated models ($\mathbf{w}^*$) i.e. $\frac{\|\mathbf{w}^{opt} - \mathbf{w}^*\|}{\|\mathbf{w}^{opt}\|}$ compared to the scikit-learn [10] solution ($\mathbf{w}^{opt}$).
- Fig. 1 shows MLlib (packaged with Spark) fails to provide optimal solutions.
- PhotonML provides very fast and accurate algorithms, but limited to AVRO data formats, which incurs additional overhead for data conversion.
- ADMM provides optimal solutions (see Fig. 1) without compromising on computational efficiency (Table 1 & 2).
- However, ADMM convergence is very sensitive to ρ updates.
- Current ADMM solution provides biggest generic repository of efficient distributed machine learning algorithms available in Python.

Ongoing work

- Comparative study of the ADMM based tool for several other multiclass algorithms.
- Extension of the generic framework to solve advanced non-convex learning formulations.

References:

[1] MLlib http://spark.apache.org/MLlib/
[2] PhotonML https://github.com/linkedin/photon-ml
[3] S. Dhar, C. Yi, N. Ramakrishnan, M. Shah, "ADMM based scalable machine learning on Spark." IEEE Big Data, 2015.
[4] M. Zinkevich et al. "Parallelized stochastic gradient descent." *Advances in Neural Information Processing Systems*. 2010.
[5] B. Recht et al. "Hogwild: A lock-free approach to parallelizing stochastic gradient descent." *Advances in Neural Information Processing Systems*. 2011.
[6] Y. Zhang, M. Jordan, "Splash: User-friendly Programming Interface for Parallelizing Stochastic Algorithms". (http://arxiv.org/abs/1506.07552).
[7] S. Boyd et. al, "Distributed optimization and statistical learning via the alternating direction method of multipliers." Foundations and Trends® in Machine Learning 3.1 (2011): 1-122
[8] S. Ohad, et. al., "Communication-Efficient Distributed Optimization using an Approximate Newton-type Method." ICML. Vol. 32. No. 1. 2014.
[9] B. McWilliams et al. "LOCO: Distributing ridge regression with random projections." (https://arxiv.org/abs/1406.3469)
[10] Scikit-Learn http://scikit-learn.org/stable/