Distributed Newton Methods for Regularized Logistic Regression

Yong Zhuang, Wei-Sheng Chin, Yu-Chin Juan, and Chih-Jen Lin

Department of Computer Science
National Taiwan University, Taipei, Taiwan
{r01922139,d01944006,r01922136,cjlin}@csie.ntu.edu.tw

Abstract. Regularized logistic regression is a very useful classification method, but for large-scale data, its distributed training has not been investigated much. In this work, we propose a distributed Newton method for training logistic regression. Many interesting techniques are discussed for reducing the communication cost and speeding up the computation. Experiments show that the proposed method is competitive with or even faster than state-of-the-art approaches such as Alternating Direction Method of Multipliers (ADMM) and Vowpal Wabbit (VW). We have released an MPI-based implementation for public use.

1 Introduction

In recent years, distributed data classification becomes popular. However, it is known that a distributed training algorithm may involve expensive communication cost between machines. The aim of this work is to construct a scalable distributed training algorithm for large-scale logistic regression.

Logistic regression is a binary classifier that has achieved a great success in many fields. Given a data set with l instances $(y_i, \boldsymbol{x}_i)$, $i = 1, \ldots, l$, where $y_i \in \{-1, 1\}$ is the label and $\boldsymbol{x}_i$ is an n-dimensional feature vector, we consider regularized logistic regression by solving the following optimization problem to obtain the model $\boldsymbol{w}$.

$$\min_{\boldsymbol{w}} \quad \tfrac{1}{2}\|\boldsymbol{w}\|^2 + C \textstyle\sum_{i=1}^{l} \log \sigma(y_i \boldsymbol{w}^T \boldsymbol{x}_i), \tag{1}$$

where $\sigma(y_i \boldsymbol{w}^T \boldsymbol{x}_i) = 1 + \exp(-y_i \boldsymbol{w}^T \boldsymbol{x}_i)$ and C is the regularization parameter.

In this paper, we design a distributed Newton method for logistic regression. Many algorithmic and implementation issues are addressed in order to reduce the communication cost and speed up the computation. For example, we investigate different ways to conduct parallel matrix-vector products and discuss data formats for storing feature values. Our resulting implementation is experimentally shown to be competitive with or faster than ADMM and VW, which were considered state-of-the-art for distributed machine learning. Recently, Lin et al. [12] implement a variant of our approach on Spark[1] through private communication, but they mainly focus on the efficient use of Spark.

This paper is organized as follows. Section 2 discusses existing approaches for distributed data classification. In Section 3, we present our implementation of a distributed Newton method. Experiments are in Section 4 while we conclude in Section 5.

[1] https://spark.apache.org/

2 Existing Methods for Distributed Classification

Many distributed algorithms have been proposed for linear classification. ADMM has recently emerged as a popular method for distributed convex optimization. Although ADMM is a known optimization method for decades, only recently some (e.g., [3, 19]) show that it is particularly suitable for distributed machine learning. Zinkevich et al. [20] proposed a way to parallelize stochastic gradient methods. Besides, parallel coordinate descent methods have been considered for linear classification (e.g., [2, 4, 14]). Agarwal et al. [1] recently report that the package VW [9] is scalable and efficient in distributed environments. VW applies a stochastic gradient method in the beginning, then switches to parallel LBFGS [13] for a faster final convergence.

In the rest of this section, we describe ADMM and VW because they are considered state-of-the-art and therefore are involved in our experiments.

2.1 ADMM for Logistic Regression

Zhang et al. [19] apply ADMM on linear support vector machine with squared hinge loss. Here we modify it for logistic regression. Assume data indices $\{1, \dots, l\}$ are partitioned to J sets $M_1, \dots, M_J$ to indicate data on J machines. We can rewrite the problem (1) to the following equivalent form.

$$\begin{aligned} \min_{\boldsymbol{w}_1,\dots,\boldsymbol{w}_J,\boldsymbol{z}} \quad & \frac{1}{2}\|\boldsymbol{z}\|^2 + C\textstyle\sum_{j=1}^{J}\sum_{i\in M_j}\log\sigma(y_i\boldsymbol{w}_j^T\boldsymbol{x}_i) \\ \text{subject to} \quad & \boldsymbol{z} = \boldsymbol{w}_j, \quad j = 1,\dots,J. \end{aligned} \tag{2}$$

All optimal $\boldsymbol{z}$, $\boldsymbol{w}_1, \dots, \boldsymbol{w}_J$ are the same as the solution of the original problem (1). ADMM repeatedly performs (3)-(5) to update primal variables $\boldsymbol{w}$ and $\boldsymbol{z}$, and Lagrangian dual variable $\boldsymbol{\mu}$ using the following rules.

$$\boldsymbol{w}_j^{k+1} = \arg\min_{\boldsymbol{w}_j} \tfrac{\rho}{2}\|\boldsymbol{w}_j - \boldsymbol{z}^k - \boldsymbol{\mu}_j^k\|^2 + C\textstyle\sum_{i\in M_j}\log\sigma(y_i\boldsymbol{w}_j^T\boldsymbol{x}_i), \tag{3}$$

$$\boldsymbol{z}^{k+1} = \arg\min_{\boldsymbol{z}} \tfrac{1}{2}\|\boldsymbol{z}\|^2 + \tfrac{\rho}{2}\textstyle\sum_{j=1}^{J}\|\boldsymbol{z} - \boldsymbol{w}_j^{k+1}\|^2 + \rho\sum_{j=1}^{J}(\boldsymbol{\mu}_j^k)^T(\boldsymbol{z} - \boldsymbol{w}_j^{k+1}), \tag{4}$$

$$\boldsymbol{\mu}_j^{k+1} = \boldsymbol{\mu}_j^{k+1} + \boldsymbol{z}^{k+1} - \boldsymbol{w}_j^{k+1}, \tag{5}$$

where $\rho > 0$ is a chosen penalty parameter. Depending on local data, the local model $\boldsymbol{w}_j$ on the jth machine can be independently updated using (3). To calculate the closed-form solution of $\boldsymbol{z}$ in (4), each machine must collect local models $\boldsymbol{w}_j, \forall j$, so an $\mathcal{O}(n)$ amount of local data from each machine is communicated across the network. The iterative procedure ensures that under some assumptions, as $k \to \infty$, $\{\boldsymbol{z}^k\}$ approaches an optimum of (1).

Recall that the communication in (4) involves $\mathcal{O}(n)$ data per machine. Obviously the cost is high for a data set with a huge number of instances (i.e., $n \gg l$). Fortunately, Boyd et al. [3] mention that splitting the data set across its features can transform the scale of the communicated data to $\mathcal{O}(l)$. For example, if we have J machines, the data matrix X is partitioned to $X_1, \dots, X_J$. Note that

$$X = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_l]^T = [X_1, \dots, X_J].$$

However, the optimization process becomes different from (2)-(5) [3].

2.2 VW for Logistic Regression

VW [1] is a machine learning package supporting distributed training. Firstly, by using only local data at each machine, it applies stochastic gradient method with adaptive learning rate [5]. Then, to get a faster convergence, VW weightedly averages the model as the initial solution for the subsequent quasi Newton method [13] on the whole data. The stage of applying stochastic gradient updates goes through all local data once for approximately solving the following sub-problem.

$$\sum_{i\in M_j}(\tfrac{1}{2}\|\boldsymbol{w}\|^2 + C\log\sigma(y_i\boldsymbol{w}^T\boldsymbol{x}_i)).$$

In the second stage, the objective function of (1) is considered and the quasi Newton method applied is LBFGS, which uses m vectors to approximate inverse Hessian (m is a user-specific parameter). To support both numerical and string feature indices in the input data, VW uses feature hashing to have a fast feature lookup. That is, it applies a hash function on the feature index to generate a new index for that feature value.

We discuss the communication cost of VW, which supports only the instance-wise data split. For the stage of running a stochastic gradient method, there is no communication until VW weightedly averages local $\boldsymbol{w}_j, \forall j$, where $\mathcal{O}(n)$ data on each machine must be aggregated. For LBFGS, it collects $\mathcal{O}(n)$ results on each machine to calculate the function value and the gradient. Therefore, the communication cost per LBFGS iteration is similar to that of each ADMM iteration under the instance-wise data split.

3 Distributed Newton Methods

In this section, we describe our proposed implementation of a distributed Newton method.

3.1 Newton Methods

We denote the objective function of (1) as $f(\boldsymbol{w})$. At each iteration, a Newton method updates the current model $\boldsymbol{w}$ by

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{s}, \tag{6}$$

where $\boldsymbol{s}$, the Newton direction, is obtained by minimizing

$$\min_{\boldsymbol{s}} \quad q(\boldsymbol{w}), \quad q(\boldsymbol{w}) = \nabla f(\boldsymbol{w})^T\boldsymbol{s} + \tfrac{1}{2}\boldsymbol{s}^T\nabla^2 f(\boldsymbol{w})\boldsymbol{s}. \tag{7}$$

Because the Hessian matrix $\nabla^2 f(\boldsymbol{w})$ is positive definite, we can solve the following linear system instead.

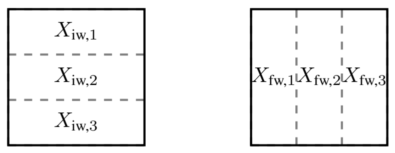
$$\nabla^2 f(\boldsymbol{w})\boldsymbol{s} = -\nabla f(\boldsymbol{w}). \tag{8}$$

For data with a huge number of features, $\nabla^2 f(\boldsymbol{w})$ becomes too large to be stored. Hessian-free methods have been developed to solve (8) without explicitly forming $\nabla^2 f(\boldsymbol{w})$. For example, Keerthi and DeCoste [8] and Lin et al. [11] apply conjugate gradient (CG) methods to solve (8). CG is an iterative procedure that requires a Hessian-vector product $\nabla^2 f(\boldsymbol{w})\boldsymbol{v}$ at each step. For logistic regression, we note that

$$\nabla^2 f(\boldsymbol{w})\boldsymbol{v} = \boldsymbol{v} + CX^T(D(X\boldsymbol{v})), \tag{9}$$

where D is a diagonal matrix with

$$D_{ii} = \frac{\sigma(y_i\boldsymbol{w}^T\boldsymbol{x}_i) - 1}{\sigma(y_i\boldsymbol{w}^T\boldsymbol{x}_i)^2}.$$



(a) Instance-wise (IW) (b) Feature-wise (FW)

Fig. 1: Two methods to distributedly store training data.

From (9), we can clearly see that a sequence of matrix-vector products is sufficient to finish the Hessian-vector product. Because $\nabla^2 f(\boldsymbol{w})$ is not explicitly formed, the memory difficulty is alleviated.

The update (6) does not guarantee the convergence to an optimum, so we apply a trust region method [11]. A constraint $\|\boldsymbol{s}\|_2 \leq \Delta$ is added to (7), where Δ is called the radius of the trust region. At each iteration, we check the ratio ρ in (10) to ensure the reduction of the function value.

$$\rho = \frac{f(\boldsymbol{w}+\boldsymbol{s}) - f(\boldsymbol{s})}{q(\boldsymbol{w})}. \tag{10}$$

If ρ is not large enough, then $\boldsymbol{s}$ is rejected and $\boldsymbol{w}$ is kept. Otherwise, $\boldsymbol{w}$ is updated by (6). Then, the radius Δ is adjusted based on ρ [11].

Although Hessian-vector products are the main computational bottleneck of the above Hessian-free approach, we note that in Newton methods other operations such as function and gradient evaluations are also important. For example, the function value in (1) requires the calculation of $X\boldsymbol{w}$. Based on this discussion, we can conclude that a scalable distributed Newton method relies on the effective parallelization of multiplying the data matrix with a vector. We will discuss more details in the rest of this section.

3.2 Instance-wise and Feature-wise Data Splits

Following the discussion in Section 2.1, we may split training data instance-wisely or feature-wisely to different machines. An illustration is in Figure 1, in which $X_{\text{iw},j}$ or $X_{\text{fw},j}$ represents the jth segment of data stored in the jth machine. We will show that the two splits lead to different communication cost. Subsequently, we use "IW/iw" and "FW/fw" to denote instance-wise and feature-wise splits.

To discuss the distributed operations easily, we use vector notation to rewrite (1) as

$$f(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\log(\sigma(YX\boldsymbol{w}))\cdot\boldsymbol{e}, \tag{11}$$

where X and Y are defined in Section 2.1, $\log(\cdot)$ and $\sigma(\cdot)$ are component-wisely applied to a vector, and "$\cdot$" stands for a dot product. In (11), $\boldsymbol{e} \in \mathbb{R}^{n\times 1}$ is a vector of all ones. Then the gradient can be represented as

$$\nabla f(\boldsymbol{w}) = \boldsymbol{w} + C(YX)^T(\sigma(YX\boldsymbol{w})^{-1} - \boldsymbol{e}). \tag{12}$$

Next we discuss details of distributed operations under the two different data splits.

Instance-wise Split: Based on (9)-(12), the distributed form of function, gradient values, and Hessian-vector products can be written as

$$f(\boldsymbol{w}) = \tfrac{1}{2}\|\boldsymbol{w}\|^2 + C\bigoplus_{j=1}^{J}\log(\sigma(Y_jX_{\text{iw},j}\boldsymbol{w}))\cdot\boldsymbol{e}_j, \tag{13}$$

$$\nabla f(\boldsymbol{w}) = \boldsymbol{w} + C \bigoplus_{j=1}^{J} (Y_j X_{\text{iw},j})^T (\sigma(Y_j X_{\text{iw},j} \boldsymbol{w})^{-1} - \boldsymbol{e}_j), \tag{14}$$

and

$$\nabla^2 f(\boldsymbol{w}) \boldsymbol{v} = \boldsymbol{v} + C \bigoplus_{j=1}^{J} X_{\text{iw},j}^T D_j X_{\text{iw},j} \boldsymbol{v}, \tag{15}$$

where Y_j, $\boldsymbol{e}_j$, and D_j are respectively the sub-matrix, the sub-vector, the sub-matrix of Y, $\boldsymbol{e}$, and D corresponding to instances in the jth machine. We use $\bigoplus$ to denote an *all-reduce* operation [15] that collects results from all machines and redistributes the sum to them. For example, $\bigoplus_{j=1}^{J} \log(\sigma(Y_j X_{\text{iw},j} \boldsymbol{w}))$ means that each machine calculates its own $\log(\sigma(Y_j X_{\text{iw},j} \boldsymbol{w}))$, and then an *all-reduce* summation is performed.
Feature-wise Split: We notice that

$$X\boldsymbol{w} = \bigoplus_{j=1}^{J} X_{\text{fw},j} \boldsymbol{w}_j, \tag{16}$$

where $\boldsymbol{w}_j$ is a sub-vector of $\boldsymbol{w}$ corresponding to features stored in the jth machine. Therefore, in contrast to IW, each machine maintains only a sub-vector of $\boldsymbol{w}$. The situation is similar to other vectors such as $\boldsymbol{s}$. However, to calculate the function value for checking the sufficient decrease (10), $\|\boldsymbol{w}_j\|^2, \forall j$ must be summed and then distributed to all machines. Therefore, the function value is calculated by

$$\frac{1}{2} \bigoplus_{j=1}^{J} \|\boldsymbol{w}_j\|^2 + C \log(\sigma(Y \bigoplus_{j=1}^{J} X_{\text{fw},j} \boldsymbol{w}_j)) \cdot \boldsymbol{e}. \tag{17}$$

Similarly, each machine must calculate part of the gradient and the Hessian-vector product:

$$\nabla f(\boldsymbol{w})_{\text{fw},p} = \boldsymbol{w}_p + C(Y X_{\text{fw},p})^T (\sigma(Y \bigoplus_{j=1}^{J} X_{\text{fw},j} \boldsymbol{w}_j)^{-1} - \boldsymbol{e}) \tag{18}$$

and

$$(\nabla^2 f(\boldsymbol{w}) \boldsymbol{v})_{\text{fw},p} = \boldsymbol{v}_p + C X_{\text{fw},p}^T D \bigoplus_{j=1}^{J} X_{\text{fw},j} \boldsymbol{v}_j,$$

where, like $\boldsymbol{w}$, only a sub-vector $\boldsymbol{v}_p$ of $\boldsymbol{v}$ is needed at the pth machine.

We notice that some *all-reduce* operations are needed for inner products in Newton methods. For example, to evaluate the value in (7) we must obtain

$$\nabla f(\boldsymbol{w})^T \boldsymbol{s} = \bigoplus_{j=1}^{J} \nabla f(\boldsymbol{w})_{\text{fw},j}^T \boldsymbol{s}_j, \tag{19}$$

where $\nabla f(\boldsymbol{w})_{\text{fw},j}$ and $\boldsymbol{s}_j$ are respectively the sub-vectors of $\nabla f(\boldsymbol{w})$ and $\boldsymbol{s}$ stored at the jth machine. The communication cost is not a concern because (19) sums J values rather than vectors in (16). Another difference from the IW approach is that the label vector $\boldsymbol{y}$ is stored in every machine because of the diagonal matrix Y in (17) and (18). It is worth mentioning that to save computational and communication cost, we can cache $\bigoplus_{j=1}^{J} X_{\text{fw},j} \boldsymbol{w}_j$ obtained in (17) for (18).
Analysis: To compare the communication cost between IW and FW, in Table 1 we show the number of operations at each machine and the amount of data sent to all others. From Table 1, to minimize the communication cost, IW and FW should be used for $l \gg n$ and $n \gg l$, respectively. We will confirm this property through experiments in Section 4.

	# of operations		# data sent to other machines	
	IW	FW	IW	FW
$f(\boldsymbol{w})$	$\mathcal{O}(\text{nnz}/J)$	$\mathcal{O}(\text{nnz}/J)$	$\mathcal{O}(1)$	$\mathcal{O}(l)$
$\nabla f(\boldsymbol{w})$	$\mathcal{O}(\text{nnz}/J)$	$\mathcal{O}(\text{nnz}/J)$	$\mathcal{O}(n)$	0
$\nabla^2 f(\boldsymbol{w})\boldsymbol{v}$	$\mathcal{O}(\text{nnz}/J)$	$\mathcal{O}(\text{nnz}/J)$	$\mathcal{O}(n)$	$\mathcal{O}(l)$
inner product	$\mathcal{O}(n)$	$\mathcal{O}(n/J)$	0	$\mathcal{O}(1)$

Table 1: A comparison between IW and FW on the amount of data distributed from one machine to all others. J is the number of machines and nnz is the number of non-zero elements in the data matrix. For FW, the calculation of $\nabla f(\boldsymbol{w})$ does not involve communication because we mentioned that $\bigoplus_{j=1}^{J} X_{\text{fw},j}\boldsymbol{w}_j$ is available while calculating $f(\boldsymbol{w})$.

3.3 Other Implementation Techniques

Load Balancing: The parallel matrix-vector product requires that all machines finish their local tasks first. To reduce the synchronization cost, we should let machines have a similar computational load. Now the computational cost is related to the number of non-zero elements, so we split data in a way such that each machine contains data of a similar number of non-zero values.

Data Format: Lin et al. [11] discuss two approaches to store the sparse data matrix X in an implementation of Newton methods: compressed sparse row (CSR) and compressed sparse column (CSC). They conclude that because of the possibility of storing a whole row or column into a higher-level memory (e.g., cache), CSR and CSC are more suitable for $l \gg n$ and $n \gg l$, respectively. Because we split data so that each machine has a similar number of non-zero elements, the number of columns/rows of the sub-matrix may vary. In our implementation, we dynamically decide the sparse format based on if the sub-matrix's number of rows is greater than columns.

Speeding Up Hessian-vector Product: Instead of sequentially calculating $X\boldsymbol{v}$, $D(X\boldsymbol{v})$, and $X^T(D(X\boldsymbol{v}))$ in (9), we can re-write $X^T DX\boldsymbol{v}$ as

$$\textstyle\sum_{i=1}^{l} D_{ii}(\boldsymbol{x}_i(\boldsymbol{x}_i^T\boldsymbol{v})). \tag{20}$$

Then the data matrix X is accessed only once rather than twice. However, this technique can only be applied for instance-wisely split data in the CSR format. If the data is split by features, then calculating each $\boldsymbol{x}_i^T\boldsymbol{v}$ needs an *all-reduce* operation. The l *all-reduce* summations in (20) cause too high communication cost in practice.

4 Experiments

In this section, we begin with describing a specific Newton method used for our implementation. Then we evaluate techniques proposed in Section 3, followed by a detailed comparison between ADMM, VW, and the distributed Newton method on objective function values and test accuracy.

4.1 Truncated Newton Method

In Section 3.1, using CG to find the direction $\boldsymbol{s}$ may struggle with too many iterations. To alleviate this problem, we consider an approach by Steihaug [16] that employs CG to approximately minimize (7) under the constraint of $\boldsymbol{s}$ being in the trust region. With the Newton direction $\boldsymbol{s}$ obtained approximately, the technique is called a truncated Newton

Data set	l	n	#nonzeros	C	Communication		Total	
					IW	FW	IW	FW
yahoo-japan	140,963	832,026	18,738,315	0.5	166.22	13.78	169.73	15.15
yahoo-korea	368,444	3,052,939	125,190,807	2	1,185.32	189.72	1,215.76	207.53
url	2,396,130	3,231,961	277,058,644	2	9,727.89	6,995.23	10,194.23	7,286.63
epsilon	500,000	2,000	1,000,000,000	2	2.16	184.03	11.37	195.26
webspam	350,000	16,609,143	1,304,697,446	32	8,909.51	159.83	9,179.42	199.36

Table 2: **Left:** The statistics of each data set. **Right:** Communication and total running time (in seconds) of the distributed Newton method using IW and FW strategies.

method. Lin et al. [11] have applied this method for logistic regression on a single machine. Here we follow their detailed settings for our distributed implementation.

4.2 Experimental Settings

1. **Data Sets:** We use five data sets listed in Table 2 for experiments, and randomly split them into 80%/20% as training set and test set respectively (an exception is epsilon because it has official training/test sets). Except yahoo-japan and yahoo-korea, all others are publicly available.[2]
2. **Platform:** We use 32 machines in a local cluster. Each machine has an 8-core processor with computing power equivalent to 8 $\times$ 1.0GHz 2007 Xeon, and 16GB memory. The network bandwidth is approximately 1Gb/s.
3. **Parameters:** For C in (1), we tried different values in $\{2^{-5}, 2^{-3}, \ldots, 2^{5}\}$, and use the one that leads to the best performance on the test set.

4.3 IW versus FW

This subsection presents a comparison between two different data splits. We run the Newton method until the following stopping condition is satisfied:

$$\|\nabla f(\boldsymbol{w})\| \leq \epsilon \frac{\min(\text{pos}, \text{neg})}{l} \|\nabla f(\mathbf{0})\|, \tag{21}$$

where $\epsilon = 10^{-6}$ and pos/neg are the number of positive and negative data.

In Table 2, we present both communication and total training time. Results are consistent with our analysis in Table 1. For example, IW is better than FW for epsilon, which has $l \gg n$. On the contrary, FW is superior for webspam because $l \ll n$. This experiment reflects that a suitable data split strategy can significantly reduce the communication cost.

By the difference between total and communication time in Table 2, we have the time for computation and synchronization. It is only similar but not the same for IW and FW because of the variance of the cluster's run-time behavior and the algorithmic differences.

4.4 Comparison Between State-of-the-art Methods on Function Values

We include the following methods for the comparison.

- **the distributed trust-region Newton method (TRON)**: It is implemented in C++ with OpenMPI [7].

[2] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets`

- **ADMM**: We implement ADMM using C++ with OpenMPI [7]. Following Zhang et al. [19], the sub-problem (3) is approximately solved by a coordinate descent method [17] with a fixed number of iterations, where the number of iterations is decided by a validation procedure. The parameter ρ in (3) follows the setting in [19].
- **VW (version 7.6)**: The package uses sockets for communications and synchronization. Because VW uses the hashing trick for the features indices, hash collisions may cause not only different training/test sets but also worse accuracy. To reduce the possibility of hash collisions, we set the size of the hashing table larger than the original data sets.[3] Although the data set may be slightly different from the original one, we will see that the conclusion of our experiments is not affected. Besides, we observe that the default setting of running a stochastic gradient method and then LBFGS is very slow on webspam, so for this problem, we apply only LBFGS.[4] Except C and the size of features hashing, we use the default parameters in VW.

In Figure 2, we compare VW, ADMM, and the distributed Newton method by checking the relation between the training time and the relative distance from the current function value to the optimum:

$$\frac{|f(\boldsymbol{w}) - f(\boldsymbol{w}^*)|}{|f(\boldsymbol{w}^*)|}.$$

The reference optimal $\boldsymbol{w}^*$ is obtained approximately by running the Newton method with a very small stopping tolerance.[5] We present results of ADMM and distributed Newton using both instance-wise and feature-wise splits. For the sparse format in Section 3.3, the distributed Newton method dynamically chooses CSC or CSR, while ADMM uses CSC because of the requirement of the coordinate descent method for the sub-problem (3).

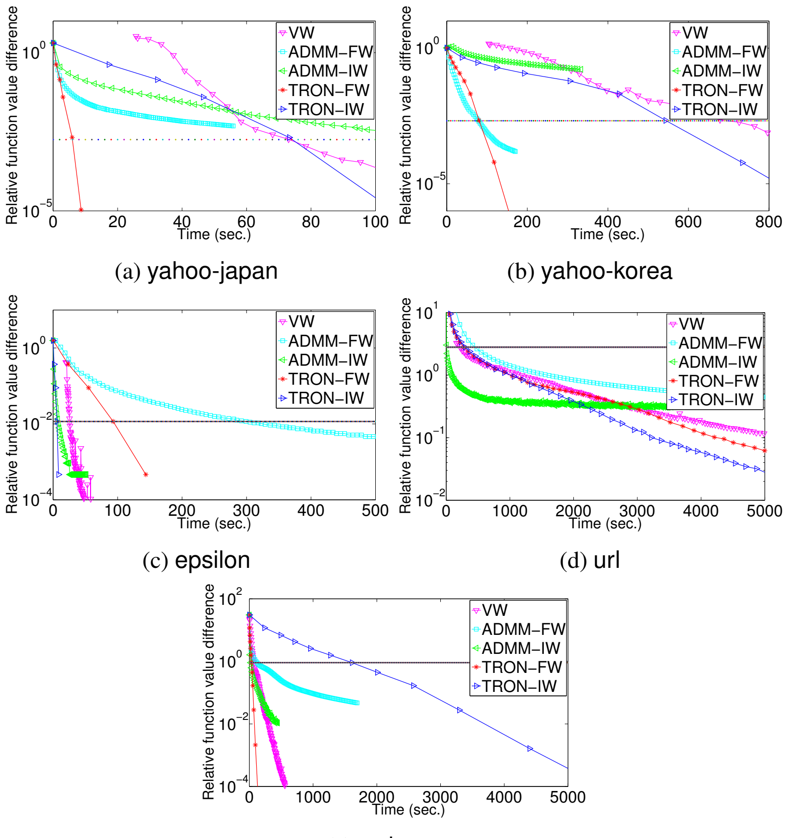
Results in Figure 2 indicate that with suitable data splits, the distributed Newton method converges faster than ADMM and VW. Regarding IW versus FW, when $l \gg n$, an instance-wise split is more suitable for the distributed Newton method, while a feature-wise split is better for $n \gg l$. This observation is consistent with Table 2. However, the same result does not hold for ADMM. One possible explanation is that regardless of data splits, the optimization processes of the distributed Newton method are exactly the same. In contrast, ADMM's optimization processes are different under IW and FW strategies [3].

In Figure 2, a horizontal line indicates that the stopping condition of (21) with $\epsilon = 0.01$ has been satisfied. This condition, used as the default condition in the Newton-method implementation of the software LIBLINEAR [6], shows that a model having similar prediction capability to the optimal solution has been obtained. In Figure 2,

[3] We select 2^{20}, 2^{22}, 2^{22}, 2^{12}, and 2^{25} as the size of the hashing table for yahoo-japan, yahoo-korea, url, epsilon, and webspam respectively.

[4] We observe that the vector $\boldsymbol{w}$ obtained after running stochastic gradient methods is a poor initial point for LBFGS to have slow convergence. It is not entirely clear what happened, so further investigation is needed.

[5] The optimal solution of VW with hashing tricks may be different from the other two methods, so we obtain its $\boldsymbol{w}^*$ separately. Because of using the relative distance, we can still compare the convergence speed of different methods.



(a) yahoo-japan (b) yahoo-korea

(c) epsilon (d) url

(e) webspam

Fig. 2: A comparison on the relative difference to the optimal objective function value. The dotted line indicates the stopping conditions (21) with $\epsilon = 0.01$ has been achieved.

ADMM can quickly reach the horizontal line for some problems, but is slow for others.

4.5 Comparison Between State-of-the-art Methods on Test Accuracy

We present in Figure 3 the relation between the training time and

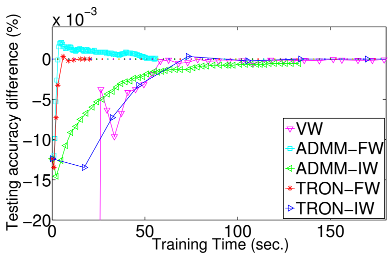
$$\frac{\text{accuracy} - \text{best\_accuracy}}{\text{best\_accuracy}},$$

where best_accuracy is the best final accuracy obtained by all methods. In the early stage ADMM is better than the other two, while the distributed Newton method gets the final stable performance more quickly on all problems except url.

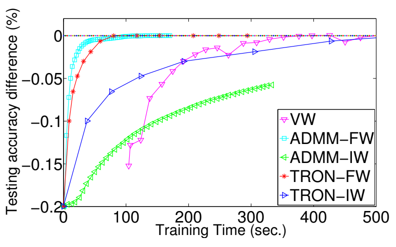
4.6 Speedup

Following Agarwal et al. [1], we compare the speedup of ADMM, VW, and the distributed Newton method for obtaining a fix test accuracy by varying the number of machines. Results are in Figure 4.

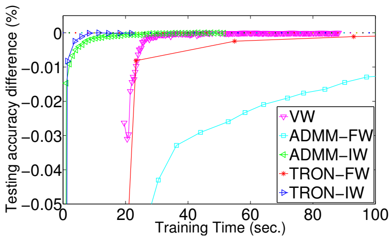
We consider the two largest sets epsilon and webspam for experiments. They have $l \gg n$ and $n \gg l$, respectively. For ADMM and the distributed Newton method, we use



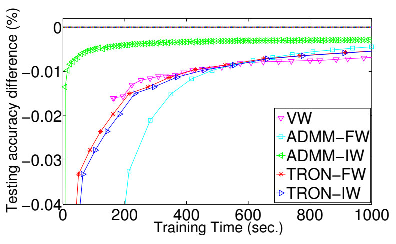
(a) yahoo-japan



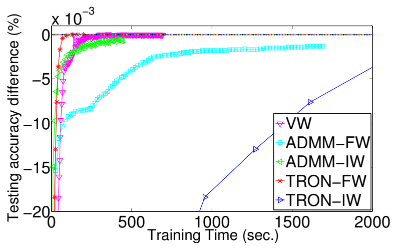
(b) yahoo-korea



(c) epsilon



(d) url



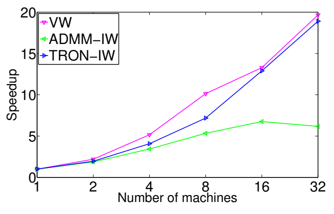
(e) webspam

Fig. 3: A test-accuracy comparison among ADMM, VW and the distributed Newton method with different data split strategies. The dotted horizontal line indicates 0 difference to the final accuracy.

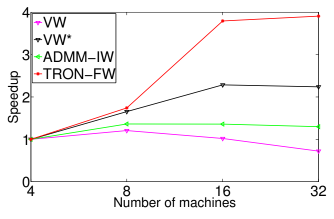
the data split strategy leading to the better convergence in Figure 2. Figure 4 shows that for epsilon ($l \gg n$), VW and the distributed Newton method yield a better speedup than ADMM as the number of machines increases, while for webspam ($n \gg l$), the distributed Newton method is better than the other two.

For the problem webspam, the speedup of VW is much worse than epsilon and problems in [1], so we conduct some investigation. For epsilon and data in [1], they have $l \gg n$. Thus we suspect that because $n \gg l$ for webspam and VW considers an instance-wise split, VW suffers from high communication cost. To confirm this point, we conduct an additional experiment in Figure 4(b). A special property of the problem webspam is that it actually has only 680,715 non-zero feature columns, although its feature indices are up to 16 million. We generate a new set by removing zero columns and rerun VW.[6] The result, indicated as VW* in Figure 4(b), clearly shows that the speedup is significantly improved.

[6] The hash size is reduced correspondingly from 2^{25} to 2^{20}.

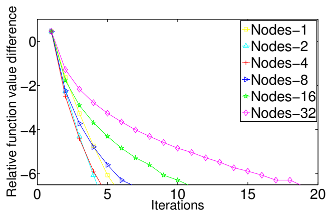


(a) epsilon

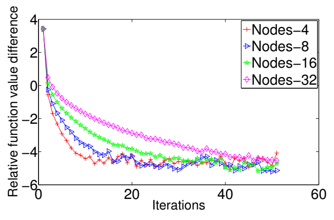


(b) webspam

Fig. 4: The speedup of different training methods.



(a) epsilon



(b) webspam

Fig. 5: The relation between the number of ADMM iterations and the decrease of the function value. We show results of using different numbers of machines.

Next, we investigate more about the unsatisfactory speedup of ADMM. Because of parallelizing only the matrix-vector products, the numbers of iterations in VW and the distributed Newton method are independent of the number of machines. In contrast, ADMM's number of iterations may significantly vary, because the reformulation in (2) is related to the number of machines. In Figure 5, we present the relation between the number of ADMM iterations and the relative difference to the optimum. It can be seen that as the number of machines increases, the higher number of iterations comes with more computational and communication costs. Thus the speedup of ADMM in Figure 4 is not satisfactory.

5 Conclusion

To the best of our knowledge, this work is the first comprehensive study on the distributed Newton method for regularized logistic regression. We carefully address important issues for distributed computation including communication and memory locality. An advantage of the proposed distributed Newton method and VW over ADMM is that the optimization processes are independent of the distributed configuration. That is, the number of iterations remains the same regardless of the number of machines. Our experiment shows that in a practical distributed environment, the distributed Newton method is faster and more scalable than ADMM and VW that are considered state-of-the-art for real-world problems.

However, because of requiring differentiability, Newton methods are more restrictive than ADMM. For example, if we consider L1-regularization by replacing $\|\boldsymbol{w}\|^2/2$

in (1) with $\|\boldsymbol{w}\|_1$, the optimization problem becomes non-differentiable, so Newton methods cannot be directly applied.[7]

Our experimental code is available at (removed for the blind review requirements).

Bibliography

1. Agarwal, A., Chapelle, O., Dudik, M., and Langford, J. (2014). A reliable effective terascale linear learning system. *JMLR*, 15:1111–1133.
2. Bian, Y., Li, X., Cao, M., and Liu, Y. (2013). Bundle CDN: a highly parallelized approach for large-scale l1-regularized logistic regression. In *ECML/PKDD*.
3. Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122.
4. Bradley, J. K., Kyrola, A., Bickson, D., and Guestrin, C. (2011). Parallel coordinate descent for l1-regularized loss minimization. In *ICML*.
5. Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12:2121–2159.
6. Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). LIBLINEAR: a library for large linear classification. *JMLR*, 9:1871–1874.
7. Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L., and Woodall, T. S. (2004). Open MPI: Goals, concept, and design of a next generation MPI implementation. In *European PVM/MPI Users' Group Meeting*, pages 97–104.
8. Keerthi, S. S. and DeCoste, D. (2005). A modified finite Newton method for fast solution of large scale linear SVMs. *JMLR*, 6:341–361.
9. Langford, J., Li, L., and Strehl, A. (2007). Vowpal Wabbit. `https://github.com/JohnLangford/vowpal_wabbit/wiki`.
10. Lin, C.-J. and Moré, J. J. (1999). Newton's method for large-scale bound constrained problems. *SIAM J. Optim.*, 9:1100–1127.
11. Lin, C.-J., Weng, R. C., and Keerthi, S. S. (2008). Trust region Newton method for large-scale logistic regression. *JMLR*, 9:627–650.
12. Lin, C.-Y., Tsai, C.-H., Lee, C.-P., and Lin, C.-J. (2014). Large-scale logistic regression and linear support vector machines using Spark. In *IEEE BigData*.
13. Liu, D. C. and Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Math. Program.*, 45(1):503–528.
14. Richtárik, P. and Takáč, M. (2012). Parallel coordinate descent methods for big data optimization. *Math. Program.* Under revision.
15. Snir, M. and Otto, S. (1998). *MPI-the complete reference: the MPI core*. MIT Press, Cambridge, MA, USA.
16. Steihaug, T. (1983). The conjugate gradient method and trust regions in large scale optimization. *SIAM J. Numer. Anal.*, 20:626–637.
17. Yu, H.-F., Huang, F.-L., and Lin, C.-J. (2011). Dual coordinate descent methods for logistic regression and maximum entropy models. *MLJ*, 85:41–75.
18. Yuan, G.-X., Chang, K.-W., Hsieh, C.-J., and Lin, C.-J. (2010). A comparison of optimization methods and software for large-scale l1-regularized linear classification. *JMLR*, 11:3183–3234.
19. Zhang, C., Lee, H., and Shin, K. G. (2012). Efficient distributed linear classification algorithms via the alternating direction method of multipliers. In *AISTATS*.
20. Zinkevich, M., Weimer, M., Smola, A., and Li, L. (2010). Parallelized stochastic gradient descent. In *NIPS*.

[7] An extension for L1-regularized problems is still possible (e.g., [10, 18]), though the algorithm becomes more complicated.